

# **AreaList Pro**

# **User Manual**

**Version 8.5**

**e-Node**  
30 rue de la République  
33150 Cenon  
France

[www.e-node.net](http://www.e-node.net)

# Copyright and Trademarks

All trade names referenced in this document are the trademark or registered trademark of their respective holders.

AreaList Pro is copyright Beckware LLC and exclusively published worldwide by e-Node.

4<sup>th</sup> Dimension, 4D Compiler, 4D, 4D Server, 4D Client, and 4D Insider are trademarks of 4D SAS.

Windows, Excel and Vista are trademarks of Microsoft Corporation.

Macintosh, MacOS and MacOS X are trademarks of Apple, Inc.

## Table of contents

<b>Copyright and Trademarks</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>About AreaList Pro</b>	<b>13</b>
Compatibility Information .....	13
Technical Support.....	13
Registration .....	14
License types .....	14
Using the AreaList Pro Manual.....	15
Cross-Referencing Format .....	15
Command List.....	15
Constant List.....	16
Command Descriptions and Syntax .....	16
<b>Installing AreaList Pro</b>	<b>17</b>
Installation: Plug-In bundle (MacOS and Windows).....	17
Backwards Compatibility .....	17
<b>Configuring AreaList Pro</b>	<b>18</b>
<b>The AreaList Pro User Interface</b>	<b>18</b>
Headers .....	19
Footers .....	20
Column Widths.....	20
Column Locking .....	21
Calculated Columns when Displaying Fields .....	21
Sorting .....	21
Enterability.....	21
Rows with Multiple Lines of Text.....	22
Color.....	22
Styles.....	23
Sorting .....	23
Scrolling.....	25
Selection.....	26
Copy to Clipboard and Edit Menu.....	26
Drag and Drop.....	26
To Drag a Row .....	26
To Drag a Column.....	27
Dragging to a Row.....	27
Dragging to a Column .....	27
To Drag a Cell .....	27

## Table of contents

Dragging to a Cell .....	27
Enterability .....	28
Initiating Data Entry .....	28
Click and Hold .....	28
Data Selection and Edit Menu Commands .....	28
Entering Data .....	28
Data Entry Using Popups .....	29
Data Entry Using Inline Controls .....	30
Moving the Current Entry Cell .....	30
Exiting Data Entry .....	30
Enterability for Fields .....	30
Resizable Windows with AreaList Pro .....	31
<b>Creating an AreaList Pro object on a form</b> .....	<b>31</b>
To configure a variable object as an AreaList Pro object .....	31
AreaList Pro Object Dimensions .....	32
Creating an Drop Area on a Form .....	32
Using the AreaList Pro Commands .....	32
Command Descriptions and Syntax .....	33
Causing an AreaList Pro Callback Method to Execute .....	33
Developer Alert .....	33
<b>Configuring AreaList Pro Using the Advanced Properties Dialog</b> .....	<b>34</b>
To Display the Advanced Properties Dialog .....	34
Setting the Data to Display .....	35
Displaying Arrays .....	35
Displaying Records .....	36
Column Enterability .....	37
Default Column .....	37
General Options .....	38
Enterability .....	38
Advanced Options .....	39
Dragging .....	39
Preview .....	40
<b>Configuring AreaList Pro Using Commands</b> .....	<b>41</b>
Using Defined Constants with AreaList Pro .....	41
Specifying the Arrays to Display .....	41
Inserting and Deleting Arrays .....	43
Modifying Array Elements Procedurally .....	43
Specifying the Fields to Display .....	43
Headers .....	44
Footers .....	44

## Table of contents

Column Widths.....	45
AreaList Pro Height .....	45
Complete Rows Display .....	45
Partial Rows Display .....	46
Column Locking .....	47
Row Height.....	47
Color.....	48
Column, Header, and Footer Colors.....	48
Row-Specific Colors.....	48
Alternate Row Colors .....	v48
Cell-Specific Colors.....	49
Miscellaneous Colors .....	49
Styles.....	50
Column, Header, and Footer Styles .....	50
Row-Specific Styles .....	50
Cell-Specific Styles .....	50
Dividing Lines .....	51
Sorting.....	51
Sort Buttons .....	51
Sort Direction Indicator.....	51
Sort Editor .....	51
Procedural Sorting.....	52
Sorting When Displaying Fields.....	52
Scrolling.....	52
Selection.....	52
Clipboard.....	53
Picture Columns .....	54
Scroll bars — Changing Displayed Form .....	54
Overview.....	54
Details: Disabling an AreaList Pro Area .....	54
Drag and Drop — Changing Form Pages .....	56
Using AreaList Pro on a Resizable Window .....	56
Performance Issues with Formatting Commands.....	56
Borders and Frames.....	57
Header/Cell Icon Support.....	57
The Escape Sentence System.....	57
Using Icons with Escape Sentences .....	58
Using Picture Library Items with Escape Sentences.....	59
Longint Reference System .....	60
Picture Objects in Headers.....	60
Commands.....	61
AL_Register (registrationKey:S) → resultCode:L.....	61

## Table of contents

%AreaListPro.....	62
AL_SetArraysNam (areaRef:L; columnNumber:I; numArrays:I; array1:S; ...; arrayN:S) → resultCode:L.....	63
AL_InsArrayNam (areaRef:L; columnNumber:I; numArrays:I; array1:S; ...; arrayN:S) → resultCode:L.....	64
AL_GetArrayNames (areaRef:L; resultArray:X; options:L) → resultCode:L .....	66
AL_RemoveArrays (areaRef:L; columnNumber:I; numArrays:I).....	66
AL_UpdateArrays (areaRef:L; updateMethod:I).....	67
AL_SetHeaders (areaRef:L; columnNumber:I; numHeaders:I; header1:S; ...; headerN:S).....	68
AL_GetHeaders (areaRef:L; headerList:X; options:L) → resultCode:L .....	69
AL_SetHeaderIcon (areaRef:L; columnNumber:I; iconAlignment:I picture:P; horPosition:I; vertPosition:I; offset:I; scaling:I).....	70
AL_SetHeaderOptions (areaRef:L; options:L; iconRef:L; callbackMethod:S) .....	72
AL_GetHeaderOptions (areaRef:L; options:L; iconRef:L; callbackMethod:S).....	73
AL_SetFooters (areaRef:L; columnNumber:I; numFooters:I; footer1:S; ...; footerN:S).....	74
AL_GetFooters (areaRef:L; footerList:X; options:L) → resultCode:L .....	74
AL_SetWidths (areaRef:L; columnNumber:I; numWidths:I; width1:I; ...; widthN:I) .....	75
AL_SetFormat (areaRef:L; columnNumber:I; format:S; columnJust:I; headerJust:I; footerJust:I; usePictHeight:I) .....	76
AL_SetDefaultFormat (selector:L; format:S).....	79
AL_GetFormat (areaRef:L; columnNumber:I; format:S; columnJust:I; headerJust:I; footerJust:I; usePictHeight:I) .....	80
AL_SetHdrStyle (areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I).....	81
AL_GetHdrStyle (areaRef:L; columnNumber:L; fontName:S; size:I; styleNum:I) .....	82
AL_SetFtrStyle (areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I) .....	82
AL_GetFtrStyle (areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I).....	83
AL_SetStyle (areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I).....	84
AL_SetDefaultStyle (selector:L; fontName:S; size:L; styleNum:L).....	85
AL_GetStyle (areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I) .....	86
AL_SetRowOpts (areaRef:L; multiRows:I; allowNoSelection:I; dragRow:I; acceptDrag:I; moveWithData:I; disableRowHighlight:I) .....	87
AL_GetRowOpts (areaRef:L; multiRows:I; allowNoSelection:I; dragRow:I; acceptDrag:I; moveWithData:I; disableRowHighlight:I) .....	89
AL_SetColOpts (areaRef:L; allowColumnResize:I; automaticResize:I; allowColumnLock:I; hideLastColumns:I; displayPixelWidth:I; dragColumn:I; acceptDrag:I).....	89
AL_GetColOpts (areaRef:L; allowColumnResize:I; automaticResize:I; allowColumnLock:I; hideLastColumns:I; displayPixelWidth:I; dragColumn:I; acceptDrag:I).....	92
AL_SetCellOpts (areaRef:L; cellSelection:I; moveWithData:I; optimization:I).....	92
AL_GetCellOpts (areaRef:L; cellSelection:I; moveWithData:I; optimization:I) .....	94
AL_SetInterface (areaRef:L; appearance:L; sortIndicator:L; useEllipsis:L; ignoreMenuMeta:L; clickDelay:L; allowPartialRow:L; useOldPopup:L; entryControls: L).....	94
AL_SetMiscOpts (areaRef:L; hideHeaders:I; areaSelected:I; postKey:S; showFooters:I; useModernLook:I).....	98
AL_GetMiscOpts (areaRef:L; hideHeaders:I; areaSelected:I; postKey:S; showFooters:I; useModernLook:I).....	99

## Table of contents

AL_SetMiscColor (areaRef:L; selector:I; alpColor:S; 4dColor:I) .....	100
AL_SetMiscRGBColor (areaRef:L; selector:L; red:L; green:L; blue:L) .....	101
AL_SetCopyOpts (areaRef:L; includeHiddenCols:I; fieldDelimiter:S; recordDelimiter:S; fieldWrapper:S) .....	101
AL_GetCopyOpts (areaRef:L; includeHiddenCols:I; fieldDelimiter:S; recordDelimiter:S; fieldWrapper:S) .....	102
AL_SetSortOpts (areaRef:L; automaticSort:I; userSort:I; allowSortEditor:I; sortEditorPrompt:S; showSortOrder:I; showSortDirIndicator:I) .....	103
AL_SetSortEditorParams (areaRef:L; windowTitle:S; prompt:S; labelList:X; columnNumberList:X) → resultCode:L .....	105
AL_GetSortEditorParams (areaRef:L; windowTitle:S; prompt:S; headerList:X; sortList:X) → resultCode:L .....	106
AL_SetSortedCols (areaRef:L; sortList:X) → resultCode:L .....	107
AL_SetForeColor (areaRef:L; columnNumber:I; alpHdrForeColor:S; 4dHdrForeColor:I; alpListForeColor:S; 4dListForeColor:I; alpFtrForeColor:S; 4dFtrForeColor:I) .....	108
AL_SetForeRGBColor (areaRef:L; columnNumber:L; hdrForeRed:L; hdrForeGreen:L; hdrForeBlue:L; listForeRed:L; listForeGreen:L; listForeBlue:L; ftrForeRed:L; ftrForeGreen:L; ftrForeBlue:L) .....	109
AL_SetBackColor (areaRef:L; columnNumber:I; alpHdrBackColor:S; 4dHdrBackColor:I; alpListBackColor:S; 4dListBackColor:I; alpFtrBackColor:S; 4dFtrBackColor:I) .....	110
AL_SetBackRGBColor (areaRef:L; columnNumber:L; hdrBackRed:L; hdrBackGreen:L; hdrBackBlue:L; listBackRed:L; listBackGreen:L; listBackBlue:L; ftrBackRed:L; ftrBackGreen:L; ftrBackBlue:L) .....	111
AL_SetDividers (areaRef:L; colDividerPattern:S; alpColDividerColor:S; 4dColDividerColor:I; rowDividerPattern:S; alpRowDividerColor:S; 4dRowDividerColor:I) .....	112
AL_SetCellBorder (areaRef:L; cellColumn:I; cellRow:L; borderLeft:I; borderTop:I; borderRight:I; borderBottom:I; offset:I; width:F; redColor:I; greenColor:I; blueColor:I) .....	114
AL_SetCellFrame (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; offset:I; width:F; redLightColor:I; greenLightColor:I; blueLightColor:I; redDarkColor:I; greenDarkColor:I; blueDarkColor:I; clearAllBorders:I) .....	115
AL_SetRGBDividers (areaRef:L; colDividerPattern:S; colDividerRed:L; colDividerGreen:L; colDividerBlue:L; rowDividerPattern:S; rowDividerRed:L; rowDividerGreen:L; rowDividerBlue:L) .....	116
AL_SetRowStyle (areaRef:L; rowNumber:L; styleNum:I; fontName:S) .....	117
AL_SetRowColor (areaRef:L; rowNumber:L; alpRowForeColor:S; 4dRowForeColor:L; alpRowBackColor:S; 4dRowBackColor:L) .....	118
AL_SetRowRGBColor (areaRef:L; rowNumber:L; rowForeRed:L; rowForeGreen:L; rowForeBlue:L; rowBackRed:L; rowBackGreen:L; rowBackBlue:L) .....	119
AL_SetAltRowColor (areaRef:L; red:L; green:L; blue:L; options:L) .....	120
AL_SetAltRowClr (areaRef:L; alpRowBackColor:S; 4dRowBackColor:I; options:L) .....	121
AL_SetCellStyle (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; styleNum:I; fontName:S) .....	122
AL_GetCellStyle (areaRef:L; cellColumn:I; cellRow:L; styleNum:I; fontName:S) .....	124
AL_SetCellColor (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; alpForeColor:S; 4dForeColor:I; alpBackColor:S; 4dBackColor:I) .....	125
AL_GetCellColor (areaRef:L; cellColumn:I; cellRow:L; 4dForeColor:I; 4dBackColor:I) .....	127
AL_SetCellRGBColor (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; cellForeRed:L; cellForeGreen:L; cellForeBlue:L; cellBackRed:L; cellBackGreen:L; cellBackBlue:L) .....	128

## Table of contents

<a href="#">AL_GetCellRGBColor</a> (areaRef:L; cellColumn:I; cellRow:L; cellForeRed:L; cellForeGreen:L; cellForeBlue:L; cellBackRed:L; cellBackGreen:L; cellBackBlue:L) .....	129
<a href="#">AL_SetCellSel</a> (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X).....	129
<a href="#">AL_SetSort</a> (areaRef:L; column1:I; ...; columnN:I) .....	131
<a href="#">AL_SetCellValue</a> (areaRef:L; row:L; column:I; alphaNumericData:S; pictData:P) .....	132
<a href="#">AL_SetLine</a> (areaRef:L; rowNumber:L) .....	132
<a href="#">AL_SetSelect</a> (areaRef:L; rowsToSelect:X) .....	133
<a href="#">AL_SetScroll</a> (areaRef:L; verticalScroll:L; horizontalScroll:I).....	134
<a href="#">AL_SetColLock</a> (areaRef:L; columns:I) .....	135
<a href="#">AL_SetHeight</a> (areaRef:L; numHeaderLines:I; headerHeightPad:I; numRowsLines:I; rowHeightPad:I; numFooterLines:I; footerHeightPad:I) .....	136
<a href="#">AL_SetMinRowHeight</a> (areaRef:L; minRowHeight:L).....	137
<a href="#">AL_SetPictureEscape</a> (areaRef:L; escapeChar:S) .....	137
<a href="#">AL_GetPictureEscape</a> (areaRef:L) → escapeChar:S .....	138

## Using the Callback Methods 139

Summary .....	139
Warnings .....	140
Executing a Callback Upon Entering an Area .....	140
Executing a Callback Upon Exiting an Area.....	141
Using Callback Methods During Data Entry.....	141
Executing a Callback Upon Entering a Cell.....	141
Executing a Callback Upon Leaving a Cell .....	142
Compatibility Note — New Menu Architecture .....	145
Compatibility Note — AL ExitCell and AL Cell deselect action become AL ExitCell and AL Cell Validate... 145	
Event Callback Interface .....	146
Edit Menu Callback .....	146
Calculated Column Callback.....	147
Commands .....	148
<a href="#">AL_SetMainCalls</a> (areaRef:L; areaEnteredMethod:S; areaExitedMethod:S) .....	148
<a href="#">AL_SetCallbacks</a> (areaRef:L; entryStartedMethod:S; entryFinishedMethod:S) .....	149
<a href="#">AL_SetEventCallback</a> (areaRef:L; callbackMethod:S; flag:L) → resultCode:L .....	151
<a href="#">AL_SetEditMenuCallback</a> (areaRef:L; callbackMethod:S) → resultCode:L.....	153
<a href="#">AL_SetCalcCall</a> (areaRef:L; columnNumber:I; calcCallback:S).....	155

## Field and Record Commands 156

Using the Field Display Capability.....	156
Temporary Arrays .....	156
Arrays and Fields.....	156
Compatibility Note — Field Display and Callbacks.....	157
Setting a Calculated Column .....	157
Setting the Callback Method .....	158
Sorting .....	158



## Table of contents

Enterability .....	158
Time Data .....	158
Displaying 4D Fields .....	159
Fields from Related One Tables .....	159
Redraw and Scrolling .....	159
Type-ahead .....	159
Copy Rows to the Clipboard .....	159
Enterability .....	159
Dragging .....	159
Sorting .....	159
Maximum Number of Records Displayed .....	160
Performance Issues When Displaying Fields .....	160
Commands .....	160
AL_SetFile (areaRef:L; tableNum:I) → resultCode:L .....	160
AL_SetFields (areaRef:L; tableNum:I; columnNumber:I; numFields:I; fieldNum1; ...; fieldNumN:I) → resultCode:L .....	161
AL_GetMode (areaRef:L) → resultCode:L .....	162
AL_GetTable (areaRef:L) → tableNumber:L .....	163
AL_GetFields (areaRef:L; tableArray:X; fieldArray:X) → resultCode:L .....	163
AL_InsertFields (areaRef:L; tableNum:I; columnNumber:I; numFields:I; fieldNum1:I ... fieldNumN:I) → resultCode:L .....	164
AL_RemoveFields (areaRef:L; columnNumber:I; numFields:I) .....	165
AL_UpdateFields (areaRef:L; updateMethod:I) .....	165
AL_SetSubSelect (areaRef:L; firstRecord:L; numRecords:L) .....	166

## Enterability 167

Initiating Data Entry .....	167
Entering Data .....	167
Filters .....	168
Click and Hold Data Entry Initiation .....	168
Entry Cell Border .....	168
Popups .....	169
Moving the Current Entry Cell .....	169
Compatibility Note — Adding or Deleting Rows from a Form Button .....	170
Redrawing the Display from the Callback Method .....	170
Exiting Data Entry .....	171
Commands .....	171
AL_SetEnterable (areaRef:L; columnNumber:I; enterability:I; popupArray:X; menuPackRef:L) .....	171
AL_SetFilter (areaRef:L; columnNumber:I; entryFilter:S) .....	173
AL_SetEntryOpts (areaRef:L; entryMode:I; allowReturn:I; displaySeconds:I; moveWithArrows:I; mapEnterKey:I; decimalCharForWin:S; useNewPopuPlcon:I) .....	174
AL_SetEntryCtrls (areaRef:L; columnNumber:I; controlType:I) .....	176

## Table of contents

AL_SetCellEnter (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; enterability:I) .....	176
AL_GetCellEnter (areaRef:L; cellColumn:I; cellRow:L; enterability:I) .....	178
AL_GetCellMod (areaRef:L) → resultCode:L .....	179
AL_GetCellValue (areaRef:L; cellRow:L; cellColumn:I; alphanumericData:T; pictData:P) .....	179
AL_SetCellHigh (areaRef:L; startPosition:I; endPosition:I) .....	180
AL_GetCellHigh (areaRef:L; startPosition:I; endPosition:I) .....	181
AL_SetCellIcon (areaRef:L; cellColumn:I; cellRow:L; pictRef:P; iconAlignment:I; horPosition:I; vertPosition:I; offset:I; scaling:I) .....	181
AL_GotoCell (areaRef:L; cellColumn:I; cellRow:L) .....	184
AL_GetCurrCell (areaRef:L; cellColumn:I; cellRow:L) .....	185
AL_GetPrevCell (areaRef:L; cellColumn:I; cellRow:L) .....	185
AL_SkipCell (areaRef:L) .....	186
AL_ExitCell (areaRef:L) .....	187

## Dragging Commands 188

Background .....	188
Technical Details of the Dragging Implementation .....	188
What are access “codes”? .....	188
After a drag .....	189
AreaList Pro on Multi-Page Layouts .....	190
Multiple Rows Dragging .....	190
Drag DataType .....	191
Drop Area .....	191
Commands .....	191
AL_SetDrgSrc (areaRef:L; sourceDataType:I; srcCode1:S; ...; srcCode10:S) .....	191
AL_SetDrgDst (areaRef:L; destDataType:I; dstCode1:S; ...; dstCode10:S) .....	192
AL_SetDrgOpts (areaRef:L; dragRowWithOptKey:I; scrollAreaSize:I; multiRowDrag:I; dragOntoRow:I) ....	193
AL_GetDrgSrcRow (areaRef:L; sourceRow:L) .....	195
AL_GetDrgSrcCol (areaRef:L; sourceCol:I) .....	196
AL_GetDrgArea (areaRef:L; destArea:L; destProcessID:I) .....	197
AL_GetDrgDstTyp (areaRef:L; destDataType:I) .....	198
AL_GetDrgDstRow (areaRef:L; destRow:L) .....	200
AL_GetDrgDstCol (areaRef:L; destCol:I) .....	200

## User Action Commands 202

AreaList Pro’s PostKey .....	202
Determining the User’s Action on an AreaList Pro Object .....	202
Example .....	203
Mouse Moved Event .....	204
Single-click and Double-click Events .....	204
Ctrl/command-click in the Column Header Event .....	205
Event Callback vs Object Method .....	205

## Table of contents

Object Methods (or Project Methods) — On Plug in Area Event .....	205
Event Callbacks .....	205
Using Both Methods .....	206
Selection .....	206
Sort Order .....	206
Column Widths .....	206
Column Information .....	206
Commands .....	207
AL_GetWidths (areaRef:L; columnNumber:I; numWidths:I; width1:I; ...; widthN:I) .....	207
AL_GetSort (areaRef:L; column1:I; ...; columnN:I) .....	208
AL_GetSortedCols (areaRef:L; sortList:X) → resultCode:L .....	209
AL_GetColumn (areaRef:L) → clickedColumn:I .....	209
AL_GetClickedRow (areaRef:L) → clickedRow:L .....	210
AL_GetSelect (areaRef:L; array:X) → resultCode:L .....	211
AL_GetCellSel (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X) → resultCode:L .....	213
AL_GetScroll (areaRef:L; verticalScroll:L; horizontalScroll:I) .....	214
AL_GetColLock (areaRef:L) → columns:I .....	214
AL_GetLine (areaRef:L) → selectedRow:L .....	215
AL_SetCellText (areaRef:L; text:T; flag:L) .....	215
AL_GetCellText (areaRef:L; text:T; flag:L) .....	216
AL_GetLastEvent (areaRef:L) → eventCode:L .....	217
<b>Utility Commands</b> .....	<b>218</b>
Drop Area .....	218
Drop Area Objects on a Multi-Page Layout .....	218
Disabling Drop Areas .....	218
Sort Editor .....	218
Area Name .....	218
Plug-in information .....	218
Commands .....	219
%AL_DropArea .....	219
AL_SetDropDst (dropAreaRef:L; dstCode1:S; ...; dstCodeN:S) .....	219
AL_ShowSortEd (areaRef:L) → sortDone:I .....	220
AL_SetAreaName (areaRef:L; areaName:S) .....	220
AL_GetAreaName (areaRef:L; areaName:S) .....	221
AL_GetVersion → version:S .....	221
AL_GetPluginPath → path:S .....	221
<b>Obsolete Commands</b> .....	<b>222</b>

## Table of contents

<b>Examples</b>	<b>223</b>
Example 1 — A Simple One-Column List .....	223
Example 2 — Displaying Headers on the List .....	226
Example 3 — Displaying Data from a Table .....	227
Example 4 — Selecting Multiple Rows .....	228
Example 5 — Allowing Data Entry .....	229
Example 6 — Restricting Data Entry to a Column .....	231
Example 7 — Validating Data Entry .....	232
Example 8 — Prohibiting Data Entry to a Specific Cell .....	234
Example 9 — Using the Event Callback Interface .....	236
Example 10 — Drag and Drop Between Areas .....	237
Enabling Drag and Drop .....	238
Introducing Generic Programming .....	238
Updating Area Entry and Exit Callback Methods .....	240
Event Callback .....	241
Handling Drag Action .....	243
Example 11 — Getting the Last Event in each Area .....	244
 <b>AreaList Pro Constant List</b>	 <b>248</b>
ALP Colors .....	248
ALP Patterns .....	248
ALP Events .....	248
ALP Entry callback actions .....	249
ALP Array commands .....	250
ALP Sort commands .....	250
ALP Column commands .....	251
ALP Row commands .....	251
ALP Entry commands .....	252
ALP Misc commands .....	253
ALP Cell commands .....	254
ALP Drag commands .....	255
ALP Field commands .....	255
ALP Appearance Constants .....	255
ALP Edit Menu Constants .....	256
ALP Format/Style Constants .....	256
 <b>AreaList Pro Command Reference — Alphabetical</b>	 <b>257</b>

# About AreaList Pro

AreaList Pro is an easy-to-use tool for implementing scrolling lists on 4<sup>th</sup> Dimension layouts. It lets you display arrays or fields.

Because AreaList Pro is a plug-in area, it is very fast, and provides capabilities not available to you using native 4D commands and objects, such as horizontal scrolling, user-resizable columns, automatic column sizing and formatting, copy to the clipboard, drag and drop interfaces, and more.

The contents of a list can even be altered directly by entering data into the AreaList Pro area using typed characters and popup menus, with full control over data entry.

Operation is extremely fast, and control objects (scroll bars, buttons, etc.) follow the MacOS and Windows interface.

AreaList Pro supports the interface standards introduced with MacOS X, WinXP and 4D 2004.

Data is passed to AreaList Pro using 4D arrays, or field numbers. If only two columns need to be displayed, create two arrays or specify two fields and pass them as parameters to AreaList Pro. No string parsing or other contortions are needed.

AreaList Pro can be used with just one command — no special formatting is required. A powerful Advanced Properties Dialog lets you configure an AreaList Pro object by simply pointing and clicking. For those cases when more control is needed, several optional commands give you complete control over the appearance of the area.

Special tools are implemented if you wish to customize the appearance and configuration of AreaList Pro, allowing the customization to be implemented rapidly.

## Compatibility Information

AreaList Pro is fully compatible with 4D/4D Server 2004 or greater (including 4D v11 SQL and 4D v12). It is compatible with MacOS and Windows clients.

## Technical Support

Technical support for AreaList Pro will be provided electronically via e-mail or our online support reporting system. You are encouraged to use the online web reporting form as it will be correctly routed to the appropriate support personnel.

[www.e-node.net](http://www.e-node.net)

### Registration

AreaList Pro requires a registration key to “unlock” the product making it a full working version. Call the **AL\_Register** command (see [AL\\_Register](#) for complete details) in the *On Startup* method.

Without the registration key, AreaList Pro will operate in demonstration mode during 20 minutes.

Version 8.3 introduced a new license design. Previous licenses will not work with this release.

In order to activate AreaList Pro 8.3 and above, you need to require a new license key from e-Node.

### License types

Like all e-Node plug-ins, AreaList Pro offers six different license types. There are no such things as MacOS vs Windows or Development vs Deployment:

- **Single user license.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Standalone or Runtime. Since the registration key is linked to a specific 4D license, you need to provide the number returned by the 4D command **GET SERIAL INFORMATION** (first parameter). A new license will be supplied for free at any time if you change your 4D version and/or get a new 4D registration key, provided that your previous licenses match the current public version at the exchange time.
- **Small server.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Server up to 10 users. The registration key is linked to your 4D Server license just as above.
- **Medium server.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Server up with 11 to 20 users. The registration key is linked to your 4D Server license just as above.
- **Large server.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Server over 20 users. The registration key is linked to your 4D Server license just as above.
- **Unlimited Single User.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on as many 4D Standalone, Runtime or Engine copies that run your 4D application(s). This is a yearly license, which expires after the date when it is to be renewed. The expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire.**
- **Unlimited OEM.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on as many 4D Server (of any number of users), 4D Standalone, Runtime or Engine copies that run your 4D application(s). This is a yearly license, which expires after the date when it is to be renewed. The expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire.**

A 4D database used to retrieve your 4D serial information is available from the following link:

<http://www.e-node.net/ftp/GetSerialInfo>

# Using the AreaList Pro Manual

General information about the AreaList Pro user interface is discussed in [The AreaList Pro User Interface](#).

An overview of the AreaList Pro commands and usage is covered in the following sections:

- [Configuring AreaList Pro Using Commands](#)
- [Field and Record Commands](#)
- [Enterability](#)
- [Dragging Commands](#)
- [User Action Commands](#)
- [Utility Commands](#)

Commands are organized by topic into individual chapters. Each chapter begins with an overview of the topic, and how to use the different commands. Each command is then covered in detail, and examples provided.

---

*Commands and parameters that are new in AreaList Pro version 8 are displayed in green characters.*

---

---

*Items that are new or modified in AreaList Pro versions 8.1 – 8.3 are displayed in pink (magenta) characters.*

---

---

*Items that are new or modified in AreaList Pro version 8.5 are displayed in orange characters.*

---

If you are unable to resolve a problem using this manual, you can contact our Technical Support Department. See [Technical Support](#).

## Cross-Referencing Format

Each time a command or section is mentioned, a cross-reference is given through hyperlinks to let you quickly find the definition for the command.

## Command List

The [alphabetical list](#) includes the parameters for each command and the page number/link to the command definition.

### Constant List

A full list of [AreaList Pro constants](#) is also available, organized by theme with each constant's actual value.

### Command Descriptions and Syntax

Each AreaList Pro command (or routine) has a syntax, or rules, that describe how to use the command in your 4D database. For each command, the name of the command is followed by the command's parameters. The parameters are enclosed in parenthesis, and separated by semicolons.

Following the command syntax description, an explanation of the command's parameters is provided. For each parameter, the type of the parameter and a description is shown. Examples are provided for each of the commands, showing the syntax as well as how the various commands are used together.

The first parameter for most commands is the long integer reference of the AreaList Pro object on the layout. This parameter is required to allow the commands to operate on the correct object.

Some routines are actually functions, which return a long integer result value. Unless otherwise indicated, the value is 0 when no error occurred, or -50 (**paramErr**) when a wrong parameter has been received.

In some instances (unlikely with the recent hardware and OS versions), AreaList Pro routines can also return memory manager errors.



# Installing AreaList Pro

This chapter outlines the steps necessary for installing AreaList Pro into your existing applications.

AreaList Pro must be installed (and de-installed) using the bundle installation method described herein.

## Installation: Plug-In bundle (MacOS & Windows)

AreaList Pro is provided as a plug-in bundle for 4D 2004, 4D v11 SQL, 4D v12 or higher.

This single version will work with MacOS and Windows deployments (you don't need separate MacOS and Windows versions).

- 1 — Locate the folder where AreaList Pro has been installed on your computer.
- 2 — Locate the 4<sup>th</sup> Dimension structure where you wish to install the AreaList Pro plug-in.
- 3 — If you don't already have a directory labeled "Plugins", create one now.
- 4 — Copy the following plug-in to your applications Plugins folder: alp.bundle.

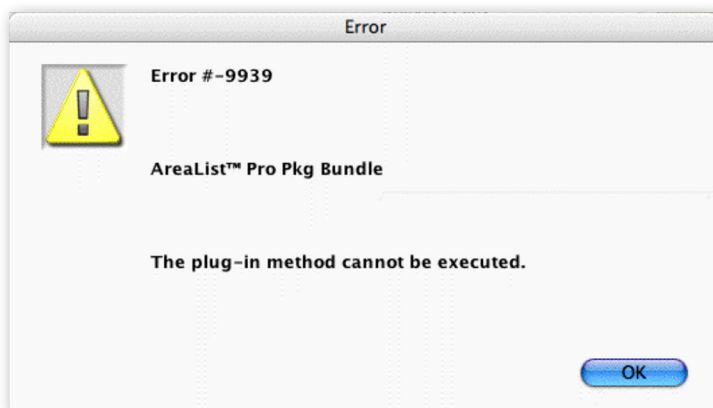
## Backwards Compatibility

If you are using AreaList Pro in an existing application, please be aware of the following changes. Failure to follow this information will result in a -9939 error (see figure below) when using AreaList Pro in heterogeneous applications (with both MacOS and Windows clients).

---

***Upgrading to AreaList Pro v8.3 or greater from pre-8.3 versions requires that you recompile your applications and you must make sure you have matching versions across Mac/Windows clients if you are using multi-platform deployments. AreaList Pro 8.3 is NOT drop in compatible, if you drop in AreaList Pro 8.3 or greater into an existing compiled application, you will receive the following error dialog.***

---



ERROR WHEN MISMATCH VERSIONS EXIST

# Configuring AreaList Pro

AreaList Pro is comprised of a suite of plug-in routines and 4<sup>th</sup> Dimension methods, designed to extend the existing 4<sup>th</sup> Dimension command set, providing a variety of miscellaneous utility routines.

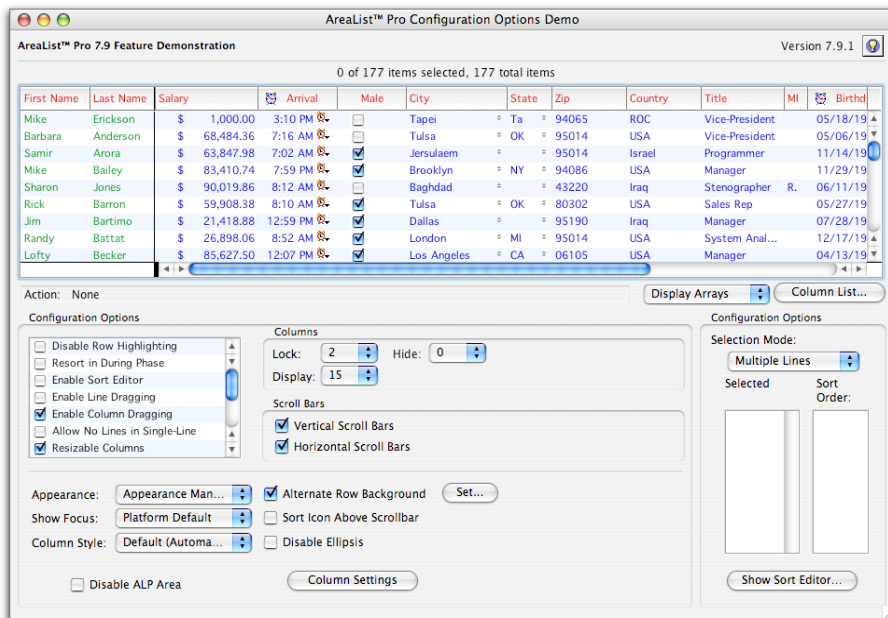
AreaList Pro Plug-In routines are routines that exist in the AreaList Pro plug-in and do not require an addition installation or configuration actions outside of standard plug-in installation.

Just make sure you have successfully registered your copy of AreaList Pro by calling the plug-in's registration routines (please see [AL\\_Register](#) for more information).

```
$ret=AL_Register("registrationKey")
```

## The AreaList Pro User Interface

AreaList Pro displays a scrolling area on 4D layouts, as shown below.



AREALIST PRO PLUG-IN AREA

AreaList Pro provides the ability to display up to 512 columns.

## Headers

Above the scrollable AreaList Pro area, there may be a row of cells called the Header area. This area is usually used to contain a description of the data displayed in each of the columns. The Header area is also used to control the sorting of the data and column dragging, if these features are enabled. The Header area is not editable by the user, and will not scroll vertically with the rest of the AreaList Pro area. See [AL\\_SetHeaders](#) and [AL\\_GetHeaders](#).

The user can click on a header to sort the list using that column. See [Sorting](#).

The user can click and drag a header to move a column to a new location. See [Drag and Drop](#).

AreaList Pro includes modern column headers, including direct platform detection. The column headers include the standard sort indication arrow (which can be enabled/disabled procedurally) to notify users which order the column is sorted.

First Name	Last Name ▼	Salary	Arrival
Todd	Zipnick	\$ 52,230.08	9:17 AM
Bob	Yuderman	\$ 22,295.00	6:24 AM
Jeffrey	Young	\$ 49,687.96	7:48 AM
Del	Yocam	\$ 63,118.86	10:16 PM
Curtis	Wright	\$ 84,651.42	1:34 AM
William	Woodward	\$ 26,602.10	8:30 AM
Ben	Wong	\$ 24,500.00	8:22 PM

MACOS X STYLE COLUMN HEADERS

First Name	Last Name ▼	Salary	Arrival
Todd	Zipnick	\$ 52,230.08	9:17 AM
Bob	Yuderman	\$ 22,295.00	6:24 AM
Jeffrey	Young	\$ 49,687.96	7:48 AM
Del	Yocam	\$ 63,118.86	10:16 PM
Curtis	Wright	\$ 84,651.42	1:34 AM
William	Woodward	\$ 26,602.10	8:30 AM
Ben	Wong	\$ 24,500.00	8:22 PM

WINXP STYLE COLUMN HEADERS

### Footers

Below the scrollable AreaList Pro area, there may be a row of cells called the Footer area. This area can be used to store information about the column, such as the total of a numeric column's data. The Footer area is not editable by the user, and will not scroll vertically with the rest of the AreaList Pro area. See [AL\\_SetFooters](#) and [AL\\_GetFooters](#).

### Column Widths

The user can resize any column by moving the arrow over the line dividing the columns in the header area. The pointer will change to the shape 4D uses in the Quick Report Editor for column resizing. Drag this column divider to resize the column.

[AL\\_SetColOpts](#) can be used to disable this feature.

---

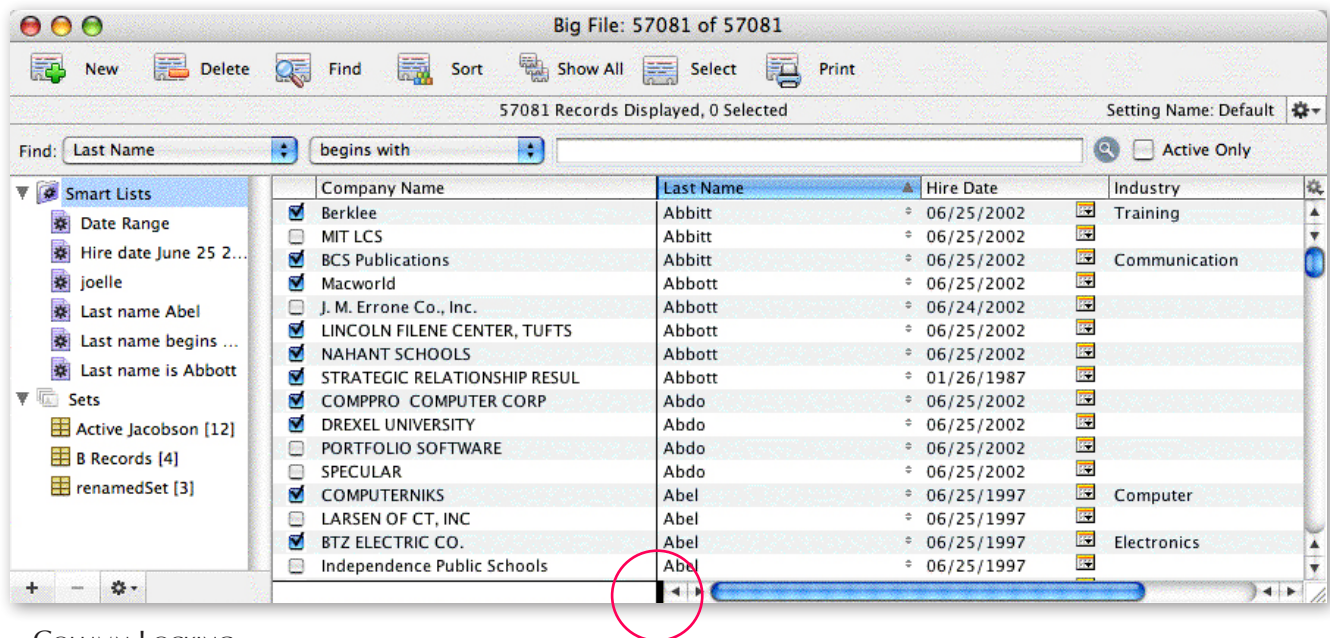
***A column cannot be resized to greater than the width of the list area minus 20 pixels.***

---

AreaList Pro will automatically truncate data and display the standard ellipsis when columns are resized smaller than the displayed data. Like the column sort icons, this setting may be procedurally enabled/disabled.

### Column Locking

One or more columns on the left side of the list can be locked in place to prevent them from scrolling horizontally. The user can adjust the lock position by dragging the Column Lock control, shown below.



COLUMN LOCKING

When columns are locked and the user clicks in the horizontal scroll bar, the locked columns will not scroll. This capability is similar to the Freeze Panes feature in Excel. When the column lock position is adjusted, the list will automatically scroll to the full left position to provide feedback to the user.

### Calculated Columns when Displaying Fields

The user interface when calculated columns are displayed is essentially the same as with fields. The few minor differences are explained below.

#### Sorting

Column headers of calculated columns will be dimmed in the [Sort Editor](#).

#### Enterability

Calculated columns will not be enterable either by typing or by using popups.

### Rows with Multiple Lines of Text

AreaList Pro allows individual rows in the list area to contain more than one line of text; however, all rows in the area will be of the same height.


### Color

AreaList Pro allows the entire range of 256 colors in the 4D palette, or the 10 colors of the built-in AreaList Pro palette. AreaList Pro also provides the ability to set the foreground and background colors using standard RGB colors.


AreaList Pro foreground colors can be applied to columns, individual rows, cells, headers, and footers. Background colors can be applied to the list area, individual rows, cells, the header area, and the footer area.

In addition, AreaList Pro provides the ability to display default row color (without having to use AreaList Pro callback to procedurally set row colors).

Using [AL\\_SetAltRowColor](#) or [AL\\_SetAltRowClr](#) routines, you can configure AreaList Pro to automatically display custom row colors, including shading rows which do not contain any information.

First Name	Last Name	Salary	 Arrival	Sex	City

MACOS X DEFAULT ROW COLORING

First Name	Last N...	Salary	 Arrival	Sex

WINXP DEFAULT ROW COLORING

### Styles

AreaList Pro supports all standard styles used by the Operating System, including Bold, Italic, Underline, Outline, Shadow, Condensed, or Extended, or any combination of these. These styles may be applied to columns, headers, footers, individual rows or cells in an AreaList Pro area.

### Sorting

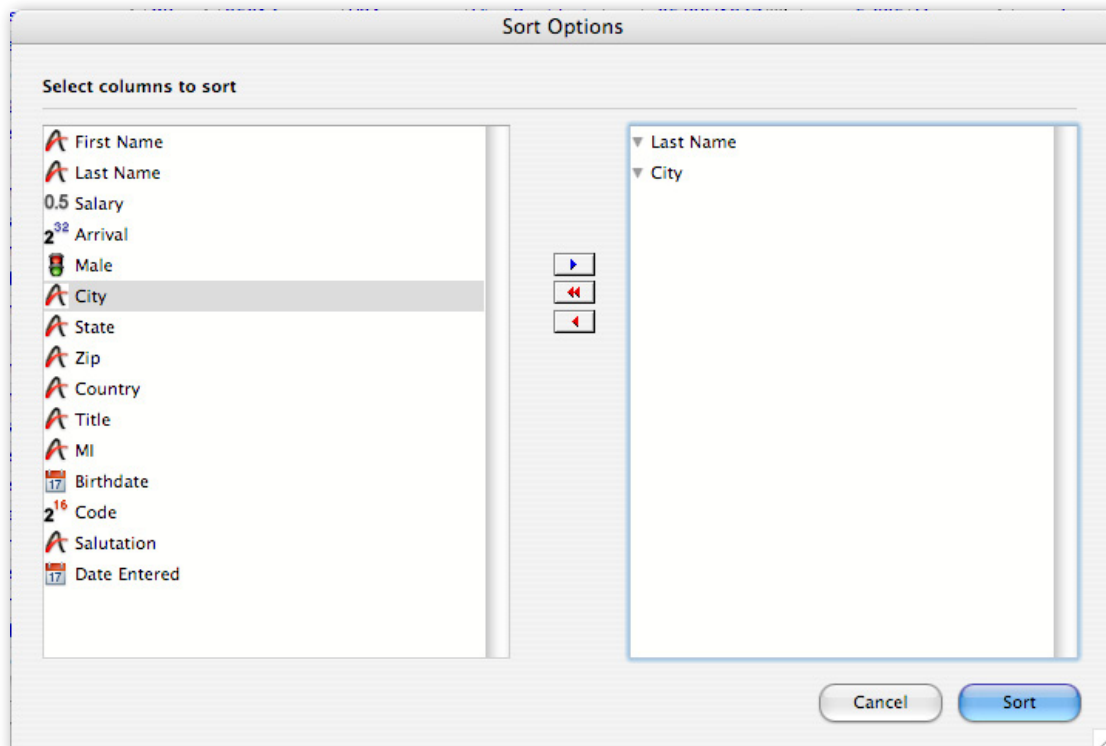
The list can be sorted in ascending (A to Z) order by clicking a column header, and sorted in descending order by option/alt-clicking the column header. Sorting the list actually sorts the 4D columns displayed in the list. If a column contains a picture, clicking its column header will cause it to highlight, but no sorting will occur.

AreaList Pro includes a sort direction indicator, which can be displayed in the upper right area.

This indicator can be clicked to change the sort order of the primary sort column. Alternately, you can create your own icon using the [AL\\_SetHeaderOptions](#) routine to override the sort direction indicator.

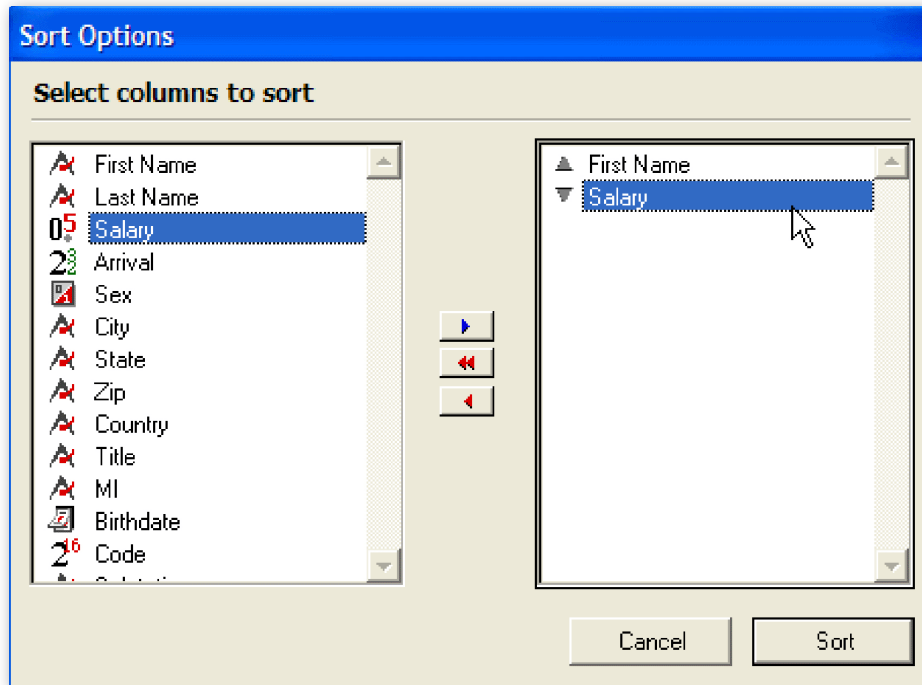
In addition to clicking a column header to sort, there is a Sort Editor available to allow sorting on multiple columns (such as **Last Name, First Name**). To access this feature, ctrl/command-click the header area of the AreaList Pro object.

The Sort Editor matches the current application platform. There are a suite of routines included in AreaList Pro which provide developers with the entry points to customize (read/write) information in the Sort Editor. The AreaList Pro default Sort Editor is displayed as a resizable window.





## The AreaList Pro User Interface



WINXP SORT EDITOR

The area on the left is a list of the columns displayed in the AreaList Pro object. An item can be added to the sort order list on the right by dragging it over the rectangle on the right or by double-clicking. To remove an item from the sort order list, drag it outside of the sort order list area.

To change the direction of the sort:

- 1 — Click the arrows to the right of each item in the sort order list (up arrow is ascending order, down arrow is descending). Although picture columns cannot be sorted, they will appear in the list of columns. However, the item(s) for the picture column(s) will be disabled and cannot be dragged into the sort order list.
- 2 — Click the Sort button.

When displaying fields, the following features are present.

- indexed fields will be bold in the Sort Editor
- fields from related one tables will be dimmed in the Sort Editor



### Scrolling

The list can be scrolled in the following ways:

- Clicking the arrows and other scroll controls.
- Using the keyboard Arrow keys. Each press of the Arrow key will scroll the list one row or column in the direction corresponding to that key. Option/alt-Arrow will scroll the list to the top, bottom, far left, or far right. AreaList Pro also has a scrolling option available when using the up/down arrows during multi-rows selections. Using the [AL\\_SetCellOpts](#) routine, you can activate the scrolling options when multiple rows are selected.
- Typing on the keyboard. As characters are typed, the current sort column will be used to vertically scroll the list. If there is a pause between typed characters, then the scrolling action will “reset”. The pause time is equal to the double-click time set in the System settings. This feature is disabled when displaying fields. See [Specifying the Fields to Display](#) for more information.
- Clicking the list area and dragging the mouse arrow outside of the list area. This action will scroll both horizontally and vertically.
- Dragging a row or column. When dragging a row or column within an AreaList Pro object, or to another valid AreaList Pro object, the destination area may scroll. See [Drag and Drop](#).

AreaList Pro provides support for live scrolling (click and dragging scrollbar will move the AreaList Pro area accordingly).

In addition, AreaList Pro includes support for wheel mouse navigating.

When an AreaList Pro area is active and you move the wheel mouse up or down, AreaList Pro will respond accordingly as if the user clicked on the scroll buttons.

When using the wheel mouse, you can scroll the AreaList Pro area horizontally by holding down the shift key.

When AreaList Pro has been configured to allow multiple rows selection, pressing the up or down Arrow keys, AreaList Pro will correctly respond and move accordingly. The following conditions have been set:

- when the up Arrow key is pressed, the row prior to the first highlighted row will be selected and will be the new active row
- when the down Arrow key is pressed, the row after the last highlighted row will be selected and will be the new active row

This interface is off by default (for backwards compatibility with previous applications) and may be activated using the **AL\_SetCellOpts** routine (second parameter, activation).

The following parameter will activate the new keyboard scrolling options when using multi-rows selection option:

**AL\_SetCellOpts**(eList;3;...) `turn on enhanced Arrow key support

### Selection

The user can create a selection in an AreaList Pro area in one of several ways: single-row, multiple rows, single cell, and multiple cells.

In single-row selection, clicking a row will select that row, and only one row can be selected at a time. In multi-rows mode, the user can select multiple rows by dragging, shift-clicking (continuous selection), or ctrl/command-clicking (discontinuous selection).

In single cell mode, clicking a cell will select only that cell, not the entire row. In multiple cells mode, the user can select none, one, or several cells. The effect of any of these methods on already selected rows or cells will be the same: the rows or cells will be deselected.

The Edit menu Select All command will select all rows when the multi-rows selection option has been enabled, or select all cells when the multiple cells selection option has been enabled.

### Copy to Clipboard and Edit Menu

Rows selected in an AreaList Pro object can be copied to the clipboard via the Edit menu Copy command.

Because of the limitations of the System clipboard (when a selection of rows are copied to the clipboard, pictures will not be copied) a blank field will appear on the clipboard where the picture would have been.

Copying rows to the clipboard will not be allowed when displaying fields. The Copy menu item will be disabled when fields are displayed.

In addition, [AL\\_SetEditMenuCallback](#) will install a callback method, which will be called when any Edit menu action occurs. See [Using the Callback Methods](#).

### Drag and Drop

The drag and drop feature of AreaList Pro allows the user to drag a row or column in an AreaList Pro object to a different position within that same area. This feature may also be used to drag a row or column to a different AreaList Pro object, to a CalendarSet object, or to an `AL_DropArea` (see [Drop Area](#)), on the same layout or a different layout.

#### To Drag a Row

AreaList Pro allows row dragging to be initiated by either option/alt-clicking on a row and dragging it, or by just dragging the row, depending on how AreaList Pro is configured. When the row is clicked on and dragged, it will move freely with the pointer.

If the row is not accepted by the destination object a rectangle will zoom back to the origin of the drag.

## The AreaList Pro User Interface

The user selects multiple rows by ctrl/command-clicking or shift-clicking. If the `dragRowWithOptKey` option of [AL\\_SetDrgOpts](#) is set to 1, then the user can also select multiple rows by dragging. Once the row(s) are selected, the user may click (or option/alt-click) to drag them. An outline of the row(s) will follow the pointer (cursor) location until the mouse is released.

You can also specify row dragging to insert between rows or drag onto rows.

### To Drag a Column

AreaList Pro allows column dragging to be initiated by clicking the column header and dragging. If the `userSort` option of [AL\\_SetSortOpts](#) is disabled, column dragging will begin immediately, and an outline of the column will appear. If user sorting is enabled, the drag begins when the mouse pointer is greater than 20 pixels to the left or right of the column, or greater than 30 pixels above or below the column header.

When the column is clicked on and dragged, it will move freely with the pointer. If the column is not accepted by the destination object a rectangle will zoom back to the origin of the drag.

### Dragging to a Row

The list will scroll when the arrow is moved within the number of pixels of the AreaList Pro object specified in [AL\\_SetDrgOpts](#). If row dragging is configured to insert rows, a small triangle will appear adjacent to the left side of the destination object, indicating the insertion position.

For dragging onto rows, the destination row will highlight.

### Dragging to a Column

The list will scroll when the arrow is moved within the number of pixels of the AreaList Pro object specified in [AL\\_SetDrgOpts](#).

A small triangle will appear adjacent to the top of the destination object, indicating the insertion position.

### To Drag a Cell

The user drags a cell by clicking upon it and dragging it.

An outline of the cell will follow the pointer (cursor) location until the mouse is released.

### Dragging to a Cell

When enabled, the user can drop an item as a row, as a column or as a cell. If the destination is a cell, an outline will be shown inside the cell that the cursor is over to indicate where the item will be dropped.

See [Dragging Commands](#) for more information.

# Enterability

## Initiating Data Entry

Data entry using typed characters may be initiated on an AreaList Pro object by several programmable methods, all of which require clicking in the cell with or without a modifier key. For example, data entry on a given cell could be initiated upon a single click in that cell, a double-click, or a double-click along with the option/alt, ctrl/command, shift, or control key.

## Click and Hold

AreaList Pro also provides the ability to initiate data entry by clicking and holding the mouse button down in the cell where you wish to perform data entry. Using this interface, users are still able to select rows via single-click and double-click. If you wish to initiate data entry using this method, use the **AL\_SetEntryOpts** routine as follows:

```
AL_SetInterface(eList;-1;-1;-1;-1;60) `initiate data entry after 1 sec (60 ticks, parameter #6) of holding  
AL_SetEntryOpts(eList;7;0) `initiate data entry via control-double-click
```

Data Entry will be initiated whenever any modifier + click action is defined as the data entry initiator (value 2..7). If the user moves the mouse during the click and hold action, AreaList Pro may interpret that as a drag action when AreaList Pro dragging actions are active

Once typed data entry is initiated, standard editing functions can be performed on the selected cell, including the Edit menu commands Cut, Copy, Paste, Clear, Select All, and Undo. This is true for cells containing pictures, also (except Select All). Alphanumeric data being edited will always appear left-justified, regardless of the column's display justification. The I-Beam pointer can be dragged across the data in the cell to select a portion or all of the data.

## Data Selection and Edit Menu Commands

In addition, the [AL\\_SetEditMenuCallback](#) routine provides the developer with a complete hook to working with the Edit menu.

## Entering Data

If string data is entered, the system beep will sound for every character typed past the maximum string length, and the typed character will be ignored (there are special programming considerations concerning this feature). If a string which exceeds the maximum string length is pasted into a cell, it will appear in the cell in its entirety, but will be truncated to the maximum string length when the insertion point leaves that cell.

Boolean data is represented during data entry by either radio buttons or a checkbox. This data may be entered via several methods, including using the space bar, using the key combinations t/f, T/F, y/n, Y/N, or the first letters, upper and lower case, of values specified in the format for the boolean data entry column.

When entering other types of data, as in 4<sup>th</sup> Dimension, data entry may be restricted to specific requirements via the use of filters.

# The AreaList Pro User Interface

## Data Entry Using Popups

AreaList Pro also has the ability to perform data entry using popup menus for column data types other than picture or boolean. Popup menus will appear as small buttons on the right side of the cell which will be labeled with a downward pointing triangle. The items contained in the popup menu represent the possible values for that cell, which are determined by you.

However, for time or date information, a special popup menu will allow the user to choose appropriate values for these data types. The presence of a popup menu in a cell does not necessarily prohibit the ability to enter typed characters.

Time and date popups can be displayed either as the “old” mode or as the “new” mode, which uses more modern fonts.

[AL\\_SetEnterable](#) sets if the column is enterable by typing, popup or both. These settings can be restricted (but not expanded) with [AL\\_SetCellEnter](#).

If the cell is enterable with popup, [AL\\_SetInterface](#) sets what kind of popup will be used (old or new.)

The time menu is shown below. To select a time, the user should begin on the left side of the popup, first selecting AM or PM, then the hour, then the minutes.

This menu will appear slightly different depending on your system settings for the time format (using a 24 hour clock, for example), but the method of selecting the time will remain basically the same.

AM	12	0
PM	1	5
	2	10
	3	15
	4	20
	5	25
	6	30
	7	35
	8	40
	9	45
	10	50
	11	55
5:40 PM		

TIME POPUP MENU (“OLD” MODE)

AM	12	0
PM	1	5
	2	10
	3	15
	4	20
	5	25
	6	30
	7	35
	8	40
	9	45
	10	50
	11	55
3:25 PM		

TIME POPUP MENU (“NEW” MODE)

The “old” date popup menu selects a date using a slightly different method: the user begins on the right side of the popup, selects the year, then month, and last, the day. Click on the arrow to scroll the years displayed on the popup. The “new” date popup is a regular calendar, with the upper left and right arrows to navigate to the previous/next month and the middle diamond to set the value to the current date.

Wed, Jun 25, 2008							January	▲
S	M	T	W	T	F	S	February	2004
1	2	3	4	5	6	7	March	2005
8	9	10	11	12	13	14	April	2006
15	16	17	18	19	20	21	May	2007
22	23	24	25	26	27	28	June	2008
29	30						July	2009
							August	2010
							September	2011
							October	2012
							November	2013
							December	▼

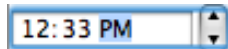
DATE POPUP MENU (“OLD” MODE)

June 2008						
S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

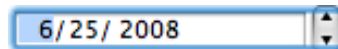
DATE POPUP MENU (“NEW” MODE)

### Data Entry Using Inline Controls

Since version 8.1, AreaList Pro allows time and/or date entry through a new interface, called Inline Controls. This option is a replacement for direct text entry.



TIME INLINE CONTROL



DATE INLINE CONTROL

[AL\\_SetEnterable](#) sets if the column is enterable by typing, popup or both. This settings can be restricted (but not expanded) with [AL\\_SetCellEnter](#).

If the cell is enterable by typing, [AL\\_SetInterface](#) sets if AreaList Pro uses plain text editing (where user can type in his date/time string) or inline date/time control.

---

*Note that the cell widths and height may need to be increased to fit the inline control interface.*

---

### Moving the Current Entry Cell

AreaList Pro speeds data entry by making it easy to move to other enterable cells once data entry is initiated. Since enterability is determined on a column by column basis, the cells adjacent to the current data entry cell may not be enterable.

AreaList Pro handles this situation by using the Tab key to move to the next enterable cell to the right. A shift-Tab combination will move data entry to the next enterable cell to the left. If there isn't an enterable cell on the same row, these key combinations will move the data entry cursor to the next or previous row, respectively.

The Return key can be used in two ways during data entry. Normally, when the Return key or shift-Return key is used, data entry will be moved to the next or previous row in the same column as the current data entry cell. However, in some cases the Return key may be used to enter a carriage return character into a text cell.

As a default, the Return key moves the data entry position. You may choose to configure the Enter key to function the same as either the Return key or the Tab key, and also have the option of causing the Arrow keys to move the insertion point from cell to cell.

### Exiting Data Entry

The user may exit data entry mode by using the mouse to click on another layout object, an AreaList Pro control or header, or a non-enterable column in the AreaList Pro area.

However, if the data that was entered was invalid, the cell cannot be exited until valid data is entered. This is determined by the entry finished callback method.

See [Using Callback Methods During Data Entry](#).

### Enterability for Fields

Columns containing fields from a related one table will not be enterable either by typing or by using popups.

### Resizable Windows with AreaList Pro

You can configure an AreaList Pro object to be resizable on a resizable window. When placed in the lower right portion of a window, AreaList Pro will draw a size box in the lower right hand corner of the window.

Click on this box and drag to resize the AreaList Pro object and its window.

## Creating an AreaList Pro Object on a Form

Implementing AreaList Pro in your 4D databases is very easy; in fact, displaying data in a AreaList Pro area can be accomplished with only one plug-in command. The AreaList Pro object is drawn on a 4D layout using the plug-in area tool.



PLUG-IN AREA TOOL

4D opens the Property List for the object, which is where the object is named and configured. The name (variable) will be used as the **areaRef** parameter for the AreaList Pro commands.

Be careful to never have two AreaList Pro objects with the same name on a 4D layout.

### To Configure a Variable Object as an AreaList Pro Object

- 1 — Create a variable object on a layout and display the variable Property List.
- 2 — Select the **AreaListPro** object type.
- 3 — Name the variable. This name will be used as the first parameter to many of the AreaList Pro commands. Note: this variable must be a process variable, not an interprocess variable (i.e. the name cannot begin with "<>" or "◇").
- 4 — The AreaList Pro object is drawn in the Layout Editor.

The first line of text contains the name of the object and its pixel dimensions, and the remaining lines are the copyright notice. If the object is small, the horizontal and vertical scroll bars are not displayed in the Layout Editor, but everything will function correctly. The display of the object name, pixel dimensions, copyright notice, and scroll bars is an indication that the object has been properly created and named.



## Creating an AreaList Pro Object on a Form

### AreaList Pro Object Dimensions

AreaList Pro provides information to allow you to properly size the AreaList Pro area and to align it with other objects on the layout in the 4<sup>th</sup> Dimension Design environment. A scale at the top of the plug-in area indicates the pixel width of the AreaList Pro object.

This may be used to align other layout objects which appear adjacent to the AreaList Pro object. Displayed next to the object's name is the width and the height of the object as it is drawn on the layout. These values include the entire area displayed by AreaList Pro, including the header and scroll bars, and they will be updated whenever the object is resized.

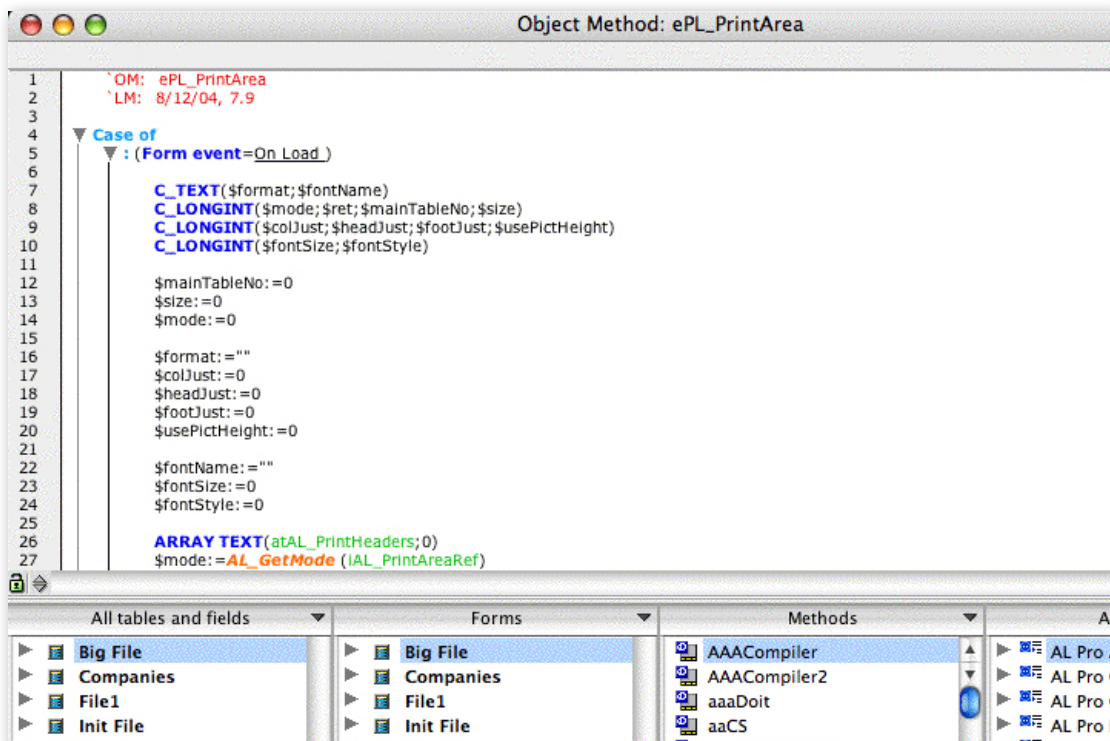
See [AreaList Pro Height](#) for additional information about controlling the height of an AreaList Pro object.

### Creating an Drop Area on a Form

To create AreaList Pro's [Drop Area](#) plug-in area, follow the same method as is used to create an AreaList Pro area, only select **AL\_DropArea** from 4D's object Property List popup. No text other than the area name will appear inside the **AL\_DropArea** object.

### Using the AreaList Pro Commands

The AreaList Pro Commands are used in the same way that a 4D command is used. Parameters are separated by the semicolon character (;). You can access the AreaList Pro commands in the method editor list. Near the bottom of the list, below the area which contains the project methods, there are several AreaList Pro command topics as shown below.





## Creating an AreaList Pro Object on a Form

Clicking on a topic presents a popup menu of the AreaList Pro commands available. Simply select a command, and 4D will enter it for you at the current cursor position.

You can also type the command directly into the method, or use the Explorer's Component page.

## Command Descriptions and Syntax

Each AreaList Pro command has a syntax, or rules, that describe how to use the command in your 4D database. For each command, the name of the command is followed by the command's parameters. The parameters are enclosed in parenthesis, and separated by semicolons. Following the command syntax description, an explanation of the command's parameters is provided. For each parameter, the type of the parameter and a description is shown. Several examples are provided, showing the syntax as well as how the various commands are used together.

The first parameter for each command is the reference (name) of the AreaList Pro object on the layout. This parameter is a long integer, and is required to allow the commands to operate on the correct object.

## Causing an AreaList Pro Callback Method to Execute

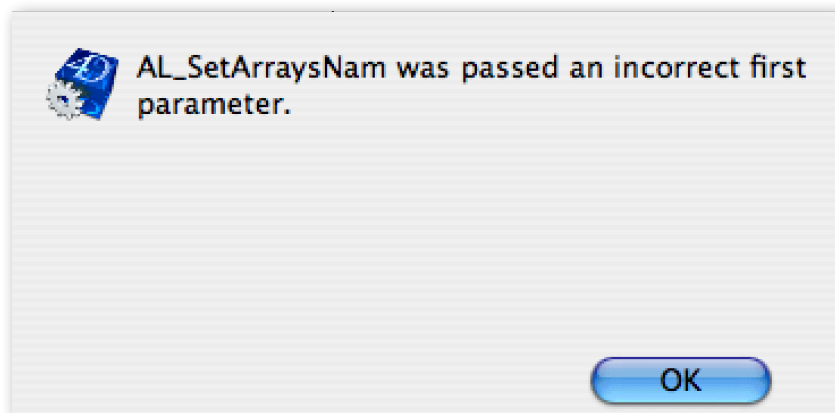
Eight callback method types are available so that the developer can react accordingly to user actions. See [Using the Callback Methods](#).

With AreaList Pro versions prior to 7.9, the events could only be managed through the object method or form method called when the user takes an action on the area, such as clicking to select a row. A new Event Callback Interface is available since version 7.9 to respond accordingly to user events.

See [Event Callback Interface](#) and [Event Callback vs Object Method](#).

## Developer Alert

If the first parameter passed to any AreaList Pro command is not the object reference, an alert box will appear, informing you of the syntax error.



INCORRECT PARAMETER ALERT

If this object reference is a PrintList Pro area or another plug-in area, AreaList Pro will also pass this information to you.

# Configuring AreaList Pro Using the Advanced Properties Dialog

AreaList Pro includes a point-and-click interface for configuring a AreaList Pro object from within the Design environment. This dialog provides access to configure nearly every feature available via AreaList Pro commands, and is very easy to use.

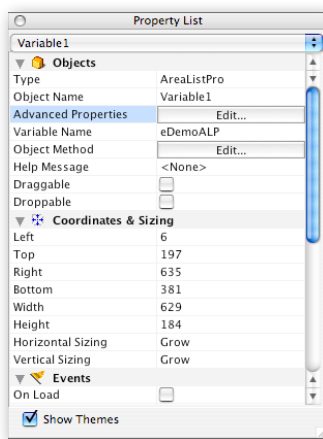
The Advanced Properties dialog lets you specify the names of the arrays to be displayed, and nearly all options. There is a preview tab to instantly view the options that you've selected.

Once you click the OK button to complete the configuration, the settings will be saved by 4D within the plug-in area object on your layout. Whenever this layout is opened in the User/Runtime environments, the settings made here will be applied to your AreaList Pro object before the form method or any object methods are executed. Essentially, you are replacing the default settings provided by AreaList Pro with new values of your choosing.

You can use commands in combination with the Advanced Properties Dialog. In this case, AreaList Pro first reads the settings specified in the dialog, then uses the settings specified by commands.

## To Display the Advanced Properties Dialog

- 1 — Double-click a AreaList Pro object in the Layout Editor. 4D will display the Object Properties palette.

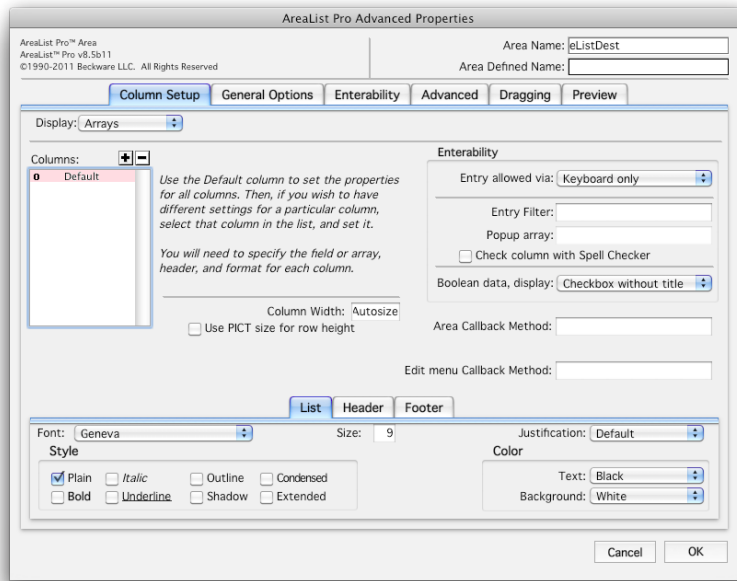


OBJECT PROPERTIES PALETTE

- 2 — Click the Advanced Properties button. The Advanced Properties Dialog will be displayed.

The dialog has several panes, accessed via the tabs at the top, which provide access to the various configuration options.

# Configuring AreaList Pro Using the Advanced Properties Dialog



ADVANCED PROPERTIES DIALOG

Once you click “OK” to complete the settings, these properties will be saved by 4D with the plug-in area object on your layout. Whenever this layout is opened in the User/Runtime environments, the settings made here will be applied to your AreaList Pro object before the layout method begins to run. Essentially, you are replacing the default settings provided by AreaList Pro with new values of your choosing.

You can specify the the arrays or fields to be displayed, and almost every other option in AreaList Pro. The only main categories that are not supported in this dialog are row and cell settings.

## Setting the Data to Display

Data is passed to AreaList Pro via 4D arrays or fields. You can tell AreaList Pro the names of the arrays or fields using the first pane on the Advanced Properties Dialog.

## Displaying Arrays

You can configure AreaList Pro to display arrays using the Advanced Properties Dialog.

You must declare all arrays in 4D before opening the form.

For example:

```
ARRAY STRING(20;aFirstName;0)
```

---

*The array must be declared before the form is opened, such as with the 4D Open window function.*

---

## Configuring AreaList Pro Using the Advanced Properties Dialog

You should usually populate the arrays in the On Load phase of the form. Then call [AL\\_UpdateArrays](#).

Example:

### Case of

: (Form event=On Load)

**SELECTION TO ARRAY** ([Table 1]First Name;aFirstName)

**AL\_UpdateArrays** (eList;-2)

### End case

The screenshot shows the 'Array Settings' dialog box. On the left, a list of columns is shown with 'First Name' selected. On the right, the 'Array Name' is 'aFirstName', 'Header Text' is 'First Name', 'Format' is empty, 'Footer Text' is empty, and 'Column Width' is 'Autosize'. There is a checkbox for 'Use PICT size for row height' which is unchecked.

ARRAY SETTINGS

## Displaying Records

You can use the Advanced Properties Dialog to display records from the database. The current selection for the Main table will be displayed at runtime.

You are responsible for establishing the selection using standard 4D commands such as **SEARCH**, **ALL RECORDS**, etc.

The project method *Compiler\_ALP* must be present in the database.

Please read the section [Temporary Arrays](#) for more information.

The screenshot shows the 'Records Settings' dialog box. On the left, a list of columns is shown with 'Default' selected. On the right, there is instructional text: 'Use the Default column to set the properties for all columns. Then, if you wish to have different settings for a particular column, select that column in the list, and set it.' and 'You will need to specify the field or array, header, and format for each column.' Below this, 'Column Width' is 'Autosize' and the 'Use PICT size for row height' checkbox is unchecked.

RECORDS SETTINGS

### Column Enterability

You can specify the enterability of each column. If you specify a popup array for a column, the array must be declared and loaded before the window is opened in the Runtime environment.

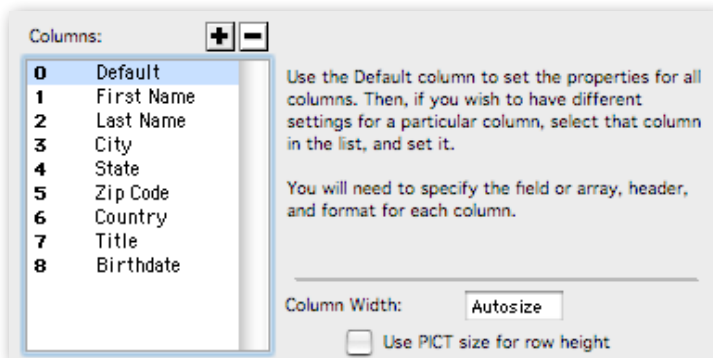
For example the following code must be executed before the **Open Window** function is called:

```
ARRAY STRING (20;aPopupFirstName;0)
```

```
SELECTION TO ARRAY ([People]First Name;aPopupFirstName)
```

See [Enterability](#) below and the [Enterability](#) section for more information about enterability configuration and options.

### Default Column



DEFAULT COLUMN

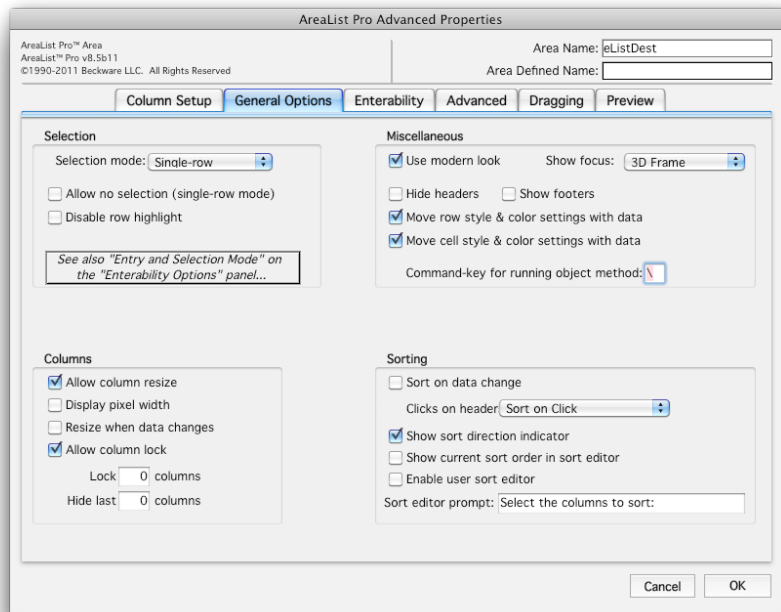
You can use the default column to setup the attributes for new columns you include by clicking the Add button. New columns that are added are assigned the settings in the default column.

This behavior is true at any time, not just the first time that the Advanced Properties dialog is configured.

If you change the settings for the default column, any new columns you add will get the settings, but existing columns will not be changed.

# Configuring AreaList Pro Using the Advanced Properties Dialog

## General Options



AreaList Pro™ Area  
AreaList™ Pro v8.5b11  
©1990-2011 Beckware LLC. All Rights Reserved

Area Name: eListDest  
Area Defined Name:

Column Setup | **General Options** | Enterability | Advanced | Dragging | Preview

**Selection**

Selection mode: Single-row

☐ Allow no selection (single-row mode)  
☐ Disable row highlight

See also "Entry and Selection Mode" on the "Enterability Options" panel...

**Miscellaneous**

☒ Use modern look    Show focus: 3D Frame

☐ Hide headers    ☐ Show footers

☒ Move row style & color settings with data  
☒ Move cell style & color settings with data

Command-key for running object method: \

**Columns**

☒ Allow column resize  
☐ Display pixel width  
☐ Resize when data changes  
☒ Allow column lock

Lock: 0 columns  
Hide last: 0 columns

**Sorting**

☐ Sort on data change  
Clicks on header: Sort on Click

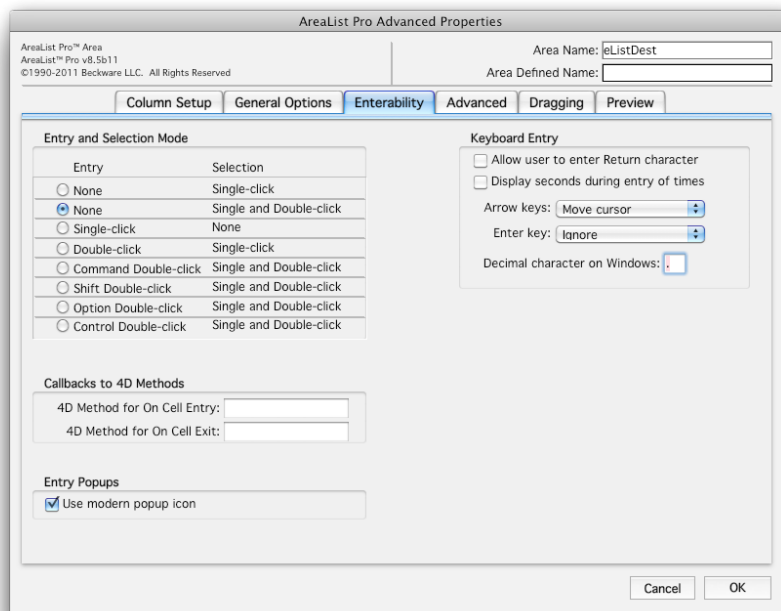
☒ Show sort direction indicator  
☐ Show current sort order in sort editor  
☐ Enable user sort editor

Sort editor prompt: Select the columns to sort:

Cancel OK

### GENERAL OPTIONS

## Enterability



AreaList Pro™ Area  
AreaList™ Pro v8.5b11  
©1990-2011 Beckware LLC. All Rights Reserved

Area Name: eListDest  
Area Defined Name:

Column Setup | General Options | **Enterability** | Advanced | Dragging | Preview

**Entry and Selection Mode**

Entry	Selection
<input type="radio"/> None	Single-click
<input checked="" type="radio"/> None	Single and Double-click
<input type="radio"/> Single-click	None
<input type="radio"/> Double-click	Single-click
<input type="radio"/> Command Double-click	Single and Double-click
<input type="radio"/> Shift Double-click	Single and Double-click
<input type="radio"/> Option Double-click	Single and Double-click
<input type="radio"/> Control Double-click	Single and Double-click

**Callbacks to 4D Methods**

4D Method for On Cell Entry:   
4D Method for On Cell Exit:

**Entry Popups**

☒ Use modern popup icon

**Keyboard Entry**

☐ Allow user to enter Return character  
☐ Display seconds during entry of times

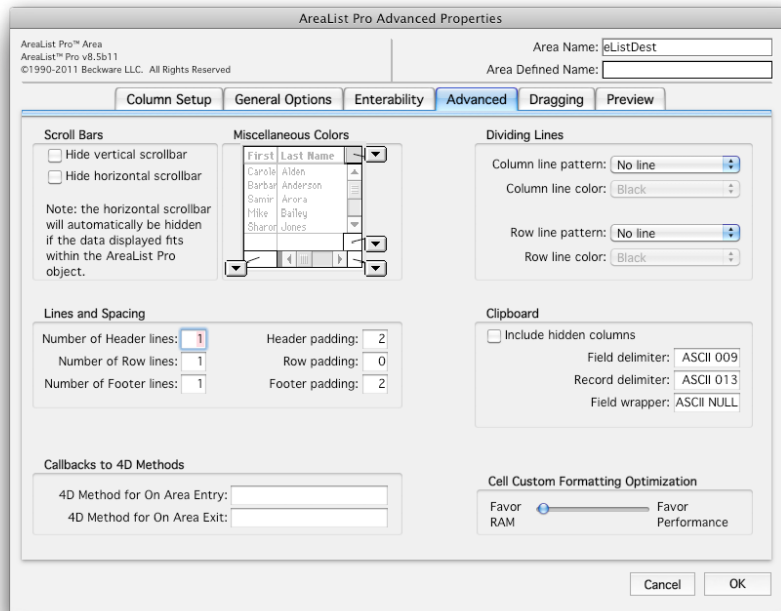
Arrow keys: Move cursor  
Enter key: Ignore  
Decimal character on Windows: .

Cancel OK

### ENTERABILITY

# Configuring AreaList Pro Using the Advanced Properties Dialog

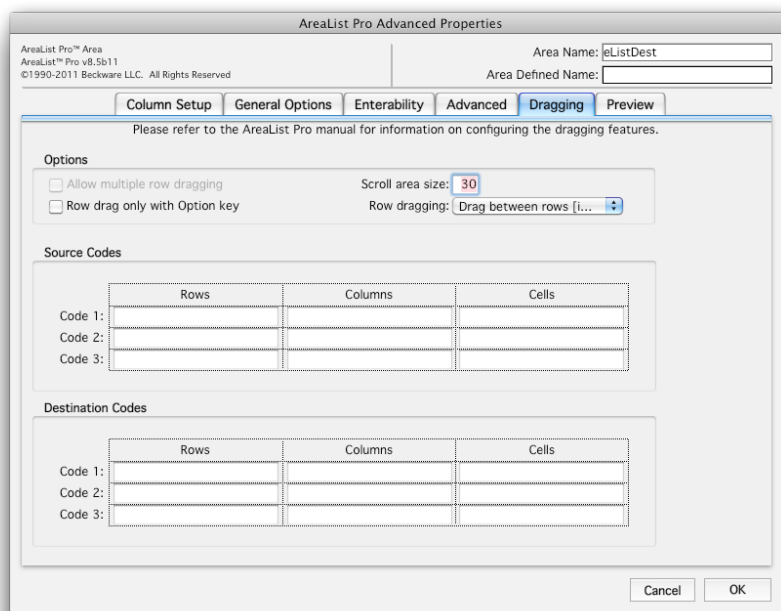
## Advanced Options



### ADVANCED OPTIONS

## Dragging

You can configure the dragging for an AreaList Pro object using this pane. Please read the section [Dragging Commands](#) for more information.



### DRAWING

# Configuring AreaList Pro Using the Advanced Properties Dialog

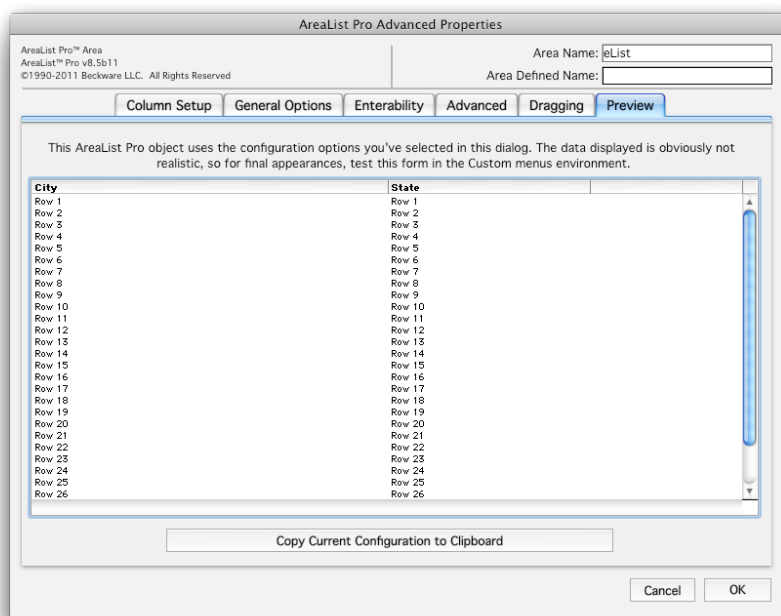
## Preview

The Preview tab allows you to preview most of the options available in the dialog.

Sample alphanumeric data is used to “fill-in” each of the columns you’ve created in the Column Setup tab.

You can quickly see the results of any configuration changes you make using this feature.

The Copy Commands for Current Configuration to Clipboard button creates 4D code and AreaList Pro commands equivalent to the current settings in the dialog, and copies it to the clipboard. This code can then be pasted into a 4D method.



PREVIEW



# Configuring AreaList Pro Using Commands

An AreaList Pro object is initialized in the On load phase as the layout is about to be displayed.

This initialization will be contained in the AreaList Pro plug-in area object method or in the form method.

## Using Defined Constants with AreaList Pro

There are [defined constants](#) that may be used as values for most of the parameters in the AreaList Pro commands. See the Constants tab of the Explorer in the 4D Design environment. These constants are categorized according to the type of command that they are associated with, such as ALP Array commands, ALP Field commands, etc.

Other constants are used for the AreaList callback interface. See [Using the Callback Methods](#).

## Specifying the Arrays to Display

4D arrays are passed to AreaList Pro for display via the [AL\\_SetArraysNam](#) and [AL\\_InsArrayNam](#) commands.

These should be performed in the On load phase of layout execution, or other form events depending upon the desired appearance of the AreaList Pro area upon initial display of the layout.

If no AreaList Pro setup is performed in the On load phase, nothing will be displayed in the space occupied by the plug-in area until setup occurs in another phase.

Whether the AreaList Pro columns are set in the On load phase or in another phase, the setup of an AreaList Pro area must follow one main rule: **AL\_SetArraysNam** or **AL\_InsArrayNam** must be called before any other AreaList Pro commands are executed.

This is necessary to provide AreaList Pro with an opportunity to allocate the data structures necessary to store formatting information for each column. These data structures are allocated on a per column basis, and **AL\_SetArraysNam** for a given column (or **AL\_InsArrayNam**) must be executed before the appearance, enterability, style, or any other property of that column can be specified.

## Configuring AreaList Pro Using Commands

If the **AL\_SetArraysNam** or **AL\_InsArrayNam** command is incorrectly used, an error code indicating the problem will be returned:

Constant	Value	Action
AL No error in arrays	0	
AL Not an array error	1	Check to make sure all arrays are correctly typed
AL Wrong type array error	2	Pointer and two-dimensional arrays are not allowed
AL Wrong number rows error	3	Make sure that all arrays have the same number of elements
AL Max arrays exceeded error	4	512 arrays is the maximum
AL Low memory array error	5	Increase 4D's RAM partition, or change your approach to use fewer or smaller arrays

### ARRAY SETUP ERROR CODES

Up to 512 arrays can be displayed by AreaList Pro, with up to fifteen columns specified in each call to **AL\_SetArraysNam** or **AL\_InsArrayNam**.

The position of the first array, **columnNumber**, and the number of arrays, **numArrays**, are also specified in these commands. All array types except for pointer and two dimensional arrays, are allowed, and all arrays must have the same number of elements.

---

***The maximum number of rows is 2,000,000,000.***

---

In addition to standard single-dimension arrays, one dimension of a two-dimensional array may be passed to **AL\_SetArraysNam** or **AL\_InsArrayNam**. For example: "My2DArray{1}" may be passed as **array1**.

While similar in purpose, the commands **AL\_SetArraysNam** and **AL\_InsArrayNam** affect previously specified arrays in different ways. In the second or any subsequent executions of **AL\_SetArraysNam**, if **columnNumber** is the number of a currently existing column, then it and any subsequent columns will be replaced by the arrays specified in the command.

However, **AL\_InsArrayNam** will actually insert the new arrays specified, and simply move existing arrays over to accommodate them. In both commands, the column number specified must either already exist or be the next higher column number available; no column numbers can be skipped.

For more information about adding, replacing and deleting arrays, read [Inserting and Deleting Arrays](#), below.

### Inserting and Deleting Arrays

After the initial setup and display of the AreaList Pro area, you may want to insert, remove, or replace arrays in the currently displayed AreaList Pro object.

To accomplish this, AreaList Pro provides the commands **AL\_InsArrayNam**, **AL\_RemoveArrays**, **AL\_SetArraysNam** and **AL\_UpdateArrays**.

These commands allow you to implement a dynamic display of data. You should keep in mind that the column number used to refer to a given column, particularly when using any of the multitude of configuration commands, may change as columns are inserted or deleted. In later attempts to configure this column, the new number must be used.

---

*No columns (arrays) should be added or deleted in a callback method. See [Using the Callback Methods](#).*

---

If new arrays of different sizes are to be displayed, then the old arrays must first be removed using **AL\_RemoveArrays**, then the new arrays added with **AL\_InsArrayNam** or **AL\_SetArraysNam**.

The [AL\\_GetArrayNames](#) routine will return an array of array names. You can only use this routine when configuring arrays via **AL\_SetArraysNam**.

### Modifying Array Elements Procedurally

When the arrays are initially specified via the **AL\_SetArraysNam** or **AL\_InsArrayNam** command, the number of array elements is established for the area.

To change the number of elements displayed in the existing arrays, new elements should be added or deleted, and the command **AL\_UpdateArrays** called with **updateMethod** set to -2.

If the value or any attribute of an array element is changed or if the number of elements is changed, but the specified arrays are the same, you should instruct AreaList Pro to refresh the area with **AL\_UpdateArrays**.

### Specifying the Fields to Display

AreaList Pro uses the **SELECTION RANGE TO ARRAY** command in 4D to get the records for display.

See [Field and Record Commands](#) for the details on fields display.

### Headers

Column headers are set with [AL\\_SetHeaders](#). This routine also provides the ability to display icons in AreaList Pro headers and cell data, using picture data contained in the cicon resources, or items stored in the 4<sup>th</sup> Dimension Picture Library.

If more than one line of text is needed in a column header, or when displaying icons, the **numHeaderLines** parameter of [AL\\_SetHeight](#) should be used. Additional space can be added to the height of a header by specifying the **headerHeightPad** parameter of this command.

Additional header attributes are specified by using [AL\\_SetHdrStyle](#), [AL\\_SetFormat](#), [AL\\_SetForeColor](#), [AL\\_SetForeRGBColor](#), [AL\\_SetBackColor](#) and [AL\\_SetBackRGBColor](#) for style, justification, foreground color, and background color, respectively.

Display of column headers can be suppressed using the **hideHeaders** parameter of [AL\\_SetMiscOpts](#).

[AL\\_SetHeaderIcon](#) provides the ability to procedurally place icons with headers.

[AL\\_SetHeaderOptions](#) provides the ability to customize the interface over the scrollbars (sort area). You can customize the icon which is displayed using “cicon” or “PICT” resource, or an item from the 4D Picture Library. See [Header/Cell Icon Support](#) for more information.

[AL\\_GetHeaders](#) will return an array of all headers for the defined AreaList Pro area.

AreaList Pro includes modern column headers, including direct platform detection. The column headers include the standard sort icon (which can be enabled/disabled procedurally) indication arrow to notify users which order the column is sorted.

### Footers

Column footers are set with [AL\\_SetFooters](#).

If more than one line of text is needed in a column footer, the **numFooterLines** parameter of [AL\\_SetHeight](#) should be used. Additional space can be added to the height of a footer by specifying the **footerHeightPad** parameter of this command.

Additional footer attributes are specified by using [AL\\_SetFtrStyle](#), [AL\\_SetFormat](#), [AL\\_SetForeColor](#), [AL\\_SetForeRGBColor](#), [AL\\_SetBackColor](#) and [AL\\_SetBackRGBColor](#) for style, justification, foreground color, and background color, respectively.

Display of column footers can be controlled using the **showFooters** parameter of the [AL\\_SetMiscOpts](#) command. Column footers are hidden by default, so you must use this command if you wish to display footers.

[AL\\_GetFooters](#) will return the footer information if you have enabled footers.

## Configuring AreaList Pro Using Commands

### Column Widths

Column widths are by default sized automatically, an option which can be overridden with [AL\\_SetWidths](#). Normally, there is no need to use this command, but for details about exceptions to this rule please read [Performance Issues with Formatting Commands](#).

Column widths can be set manually by using [AL\\_SetWidths](#); however, you may want to view the widths generated by AreaList Pro's automatic column sizing as a good starting reference. The `displayPixelWidth` parameter of [AL\\_SetColOpts](#) should be set to 1 to enable this feature, which allows you to toggle between the header text and the column width by clicking on the check box that appears in the bottom right corner of the AreaList Pro object. Additionally, the columns can be resized in the Runtime environment, and the column width values are updated immediately.

When using this feature, you should be sure to enable the display of headers by passing 0 in the `hideHeaders` parameter of [AL\\_SetMiscOpts](#).

[AL\\_GetWidths](#) will return displayed column widths.

[AL\\_GetColOpts](#) will return the current settings configured using [AL\\_SetColOpts](#).

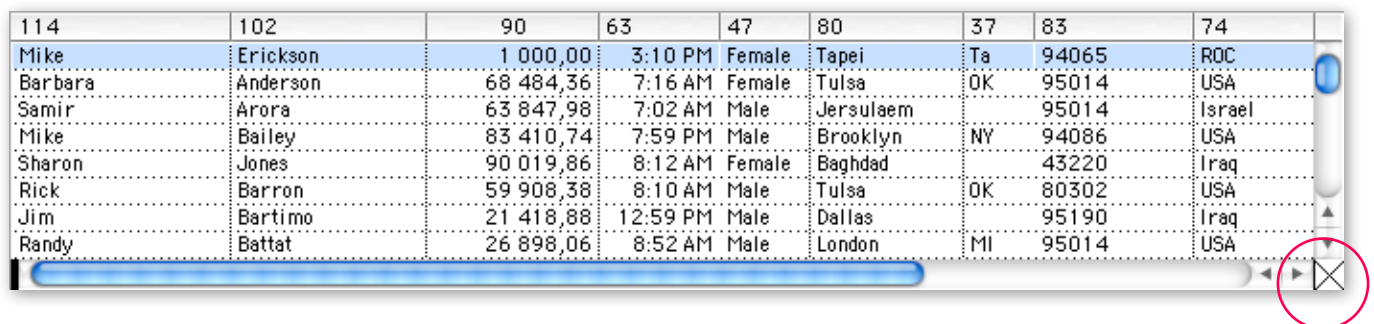
### AreaList Pro Height

#### Complete Rows Display

Whenever an array or field command is called, AreaList Pro performs calculations necessary to size the plug-in area based on the size of the object as drawn on the layout. AreaList Pro will always ensure that only complete rows are displayed in the AreaList Pro area.

However, this means that the actual height of the plug-in area as displayed in the User or Runtime environment may be slightly less than the height in the Layout Editor. This can be a hindrance when you are attempting to align other layout objects with the AreaList Pro object.

To ensure that the AreaList Pro object does not change its size when displayed in the Runtime environment, a tool is available to tell you what size to make the area. To use this tool, first set the `displayPixelWidth` parameter of [AL\\_SetColOpts](#) to 1, then click the checkbox as shown.



114	102	90	63	47	80	37	83	74
Mike	Erickson	1 000,00	3:10 PM	Female	Tapei	Ta	94065	ROC
Barbara	Anderson	68 484,36	7:16 AM	Female	Tulsa	OK	95014	USA
Samir	Arora	63 847,98	7:02 AM	Male	Jersulaem		95014	Israel
Mike	Bailey	83 410,74	7:59 PM	Male	Brooklyn	NY	94086	USA
Sharon	Jones	90 019,86	8:12 AM	Female	Baghdad		43220	Iraq
Rick	Barron	59 908,38	8:10 AM	Male	Tulsa	OK	80302	USA
Jim	Bartimo	21 418,88	12:59 PM	Male	Dallas		95190	Iraq
Randy	Battat	26 898,06	8:52 AM	Male	London	MI	95014	USA

COLUMN WIDTHS/HEADERS TOGGLE

The mouse pointer will change from an arrow to a pixel count whenever it is over the list and this option is set.

## Configuring AreaList Pro Using Commands

When clicked on a row, this counter will display the necessary height of the AreaList Pro object for that row to be the bottom row displayed.

For example, if ten rows are displayed in the area, and you click the seventh row, the number displayed by the pointer will be the height of the object necessary to display exactly seven rows. You can then size the AreaList Pro object in the Design environment using the displayed height.

Please read the section [AreaList Pro Object Dimensions](#) for more information.

The header size, footer size and the horizontal scroll bar will be taken into account if they are displayed.

---

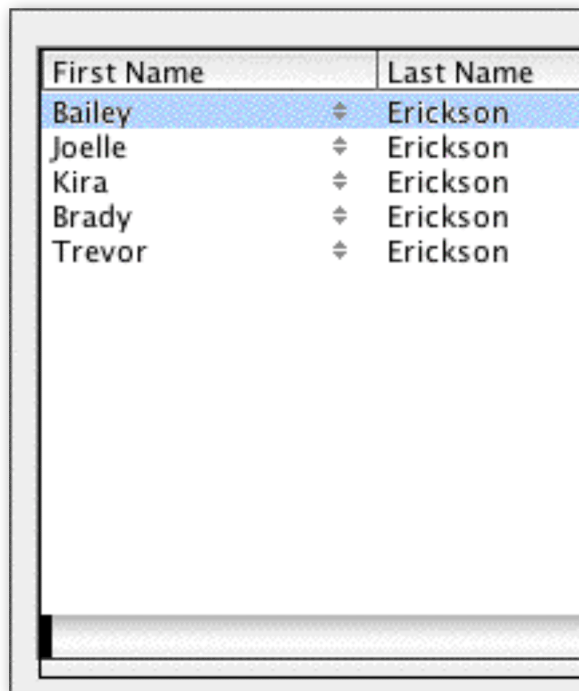
***This feature is unavailable if enterability can be initiated with a single click.***

---

## Partial Rows Display

Alternately, there is an option whereby you can configure AreaList Pro areas to draw in the exact same height as defined in the form. With the settings above, AreaList Pro will resize the height of the area to match the font attributes and assure that only a complete row was visible.

This has an adverse effect that the area may be resized to be shorter than that which was drawn in the form, causing some user interface inconsistencies.

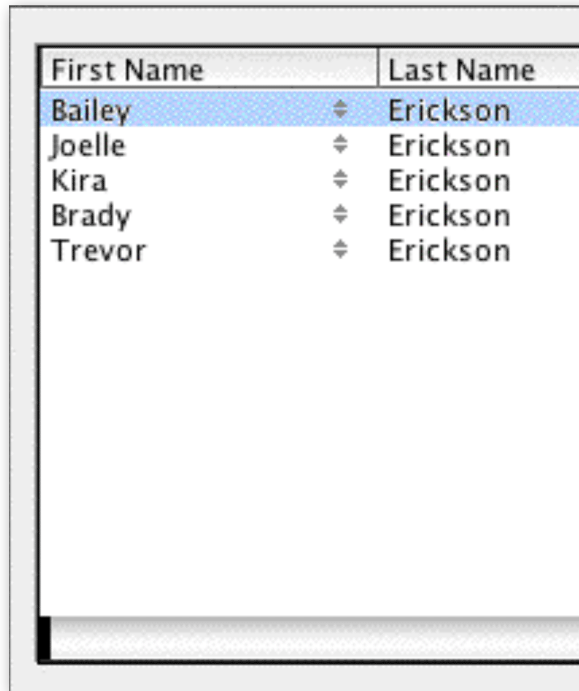


First Name		Last Name
Bailey	↕	Erickson
Joelle	↕	Erickson
Kira	↕	Erickson
Brady	↕	Erickson
Trevor	↕	Erickson

AREALIST PRO AREA AUTO SIZED (SEE GAP AT BOTTOM)

## Configuring AreaList Pro Using Commands

This interface can be controlled using the `allowPartialRow` parameter in [AL\\_SetInterface](#):



First Name		Last Name
Bailey	↕	Erickson
Joelle	↕	Erickson
Kira	↕	Erickson
Brady	↕	Erickson
Trevor	↕	Erickson

AREALIST PRO DRAWN IN ACTUAL SIZE OF FORM OBJECT

## Column Locking

You can set the lock position using [AL\\_SetColLock](#). [AL\\_GetColLock](#) returns the current position of the column lock. You can also disable the column lock control by using the `allowColumnLock` parameter of [AL\\_SetColOpts](#).

[AL\\_GetColOpts](#) will return the current settings configured using [AL\\_SetColOpts](#).

## Row Height

Row height is determined by a combination of the height of the text line or picture, the number of lines (multiple lines of text), and any additional padded space. The height of each line of text is determined by the font and point size selected, which are set with [AL\\_SetStyle](#).

The number of text lines and the amount of padding are set with the `numRowLines` and `rowHeightPad` parameters of [AL\\_SetHeight](#). Padded space is the amount of space above and below the text block, (half of the amount above, half below). All rows will be of the same height.

[AL\\_SetMinRowHeight](#) provides the ability to set the minimum row height for AreaList Pro rows. This is different than row padding as it will allow you to set individual rows to appear with extra white space, regardless of the amount of data.



# Color

## Column, Header, and Footer Colors

Foreground and background colors can be specified for an AreaList Pro object using [AL\\_SetForeColor](#), [AL\\_SetForeColorRGBColor](#), [AL\\_SetBackColor](#) and [AL\\_SetBackColorRGBColor](#). The foreground and background colors can be specified for each column, column header, and column footer.

## Row-Specific Colors

[AL\\_SetRowColor](#) and [AL\\_SetRowRGBColor](#) are used to set the foreground and background colors of a specified row, and will override any column specification. You can revert to the original column settings with [AL\\_SetRowColor](#) by setting the `alpRowForeColor` or `alpRowBackColor` parameter to the empty string (""), and the `4dRowForeColor` or `4dRowBackColor` parameter to -1. Use this command to override all row-specific color settings by passing 0 for the `rowNumber` parameter.

By default, the row color will move with a row if the columns are sorted or a row is dragged. This can be overridden using the `moveWithData` parameter of [AL\\_SetRowOpts](#).

## Alternate Row Colors

AreaList Pro provides the ability to display default row color without any additional programming such as callback routines.

Using [AL\\_SetAltRowColor](#) or [AL\\_SetAltRowClr](#), you can configure AreaList Pro to automatically display custom row colors, including shading rows which do not contain any information.

First Name	Last Name	Salary	Arrival	Sex	City

MACOS X DEFAULT ROW COLORING

First Name	Last N...	Salary	Arrival	Sex

WINXP DEFAULT ROW COLORING



## Configuring AreaList Pro Using Commands

[AL\\_SetAltRowColor](#) provides the ability to set the alternate row colors for an AreaList Pro area. The colors are defined using a standard RGB pattern and can optionally be configured to display the alternate row color in blank rows to fill the entire area with a consistent interface.

You may optionally display the alternate row color for odd and/or even rows, including empty rows (those below the last row).

[AL\\_SetAltRowClr](#) performs the same action as ***AL\_SetAltRowColor***, except that it uses the standard AreaList Pro color formatting parameters as routines such as [AL\\_SetMiscColor](#).

In addition, [AL\\_SetRowColor](#) provides a simple method of setting the alternate row colors for an AreaList Pro area. The row colors used will be determined based on the current platform (OS9, OSX, WinXP or Vista).

### Cell-Specific Colors

Individual column elements, called cells, can be assigned a unique foreground color and background color.

This capability can be used to set negative numbers in red, provide special formatting to show the current selected or enterable cell, and design more attractive and useful lists.

These attributes can be set in the On load phase, and either of the AreaList Pro callback methods (see [Using the Callback Methods](#)).

You can use [AL\\_SetCellColor](#) or [AL\\_SetCellRGBColor](#) to set the color configuration for an individual cell, a range of cells, or a selection of discontinuous cells.

[AL\\_GetCellColor](#) and [AL\\_GetCellRGBColor](#) are used to determine any cell-specific colors for a particular cell. ***AL\_GetCellColor*** can only determine a color which has been set using the 4D palette of 256 colors, not the AreaList Pro palette.

Use the `moveWithData` option of [AL\\_SetCellOpts](#) to keep the cell-specific information with a cell when a row or column is dragged to a new location or the list is sorted.

### Miscellaneous Colors

You can use [AL\\_SetMiscColor](#) or [AL\\_SetMiscRGBColor](#) to set the color of four different areas of an AreaList Pro object.

These areas are the upper right, lower right, to the right of the footer (if displayed), and the lower left if the column lock has locked one or more columns.

# Styles

## Column, Header, and Footer Styles

Styles for displayed columns can be set on a column by column basis using [AL\\_SetStyle](#) to set the style for the data, [AL\\_SetHdrStyle](#) to set the header style, and [AL\\_SetFtrStyle](#) to set the footer style. If a 0 is used in the `columnNumber` parameter, the style will be applied to all columns.

[AL\\_GetStyle](#) returns the formatting options set using the **AL\_SetStyle** routine.

[AL\\_GetHdrStyle](#) returns the formatting options set using the **AL\_SetHdrStyle** routine.

[AL\\_GetFtrStyle](#) returns the formatting options set using the **AL\_SetFtrStyle** routine.

In addition, [AL\\_SetDefaultStyle](#) can be used to set the default values for the list data, the headers and the footers of all AreaList Pro areas.

## Row-Specific Styles

[AL\\_SetRowStyle](#) is used to set the font and style of a specified row, and will override any column specification. You can revert to the original column settings by setting the `styleNum` parameter to -1.

Use this command to override all row-specific style settings by passing 0 for the `rowNumber` parameter.

By default, the row style will move with a row if the columns are sorted or a row is dragged.

This can be overridden using the `moveWithData` parameter of [AL\\_SetRowOpts](#).

[AL\\_GetRowOpts](#) will return the current settings configured using **AL\_SetRowOpts**.

## Cell-Specific Styles

Individual column elements, called cells, can be assigned a unique font and style.

This capability can be used to provide special formatting to show the current selected or enterable cell, and design more attractive and useful lists.

These attributes can be set in the On load phase, and either of the AreaList Pro callback methods (see [Using the Callback Methods](#)).

You can use [AL\\_SetCellStyle](#) to set the font and style configuration for an individual cell, a range of cells, or a selection of discontinuous cells. [AL\\_GetCellStyle](#) is used to determine any cell-specific formats for a particular cell.

Use the `moveWithData` option of [AL\\_SetCellOpts](#) to keep the cell-specific information with a cell when a row or column is dragged to a new location or the list is sorted.

[AL\\_GetCellOpts](#) will return the current settings configured using **AL\_SetCellOpts**.

### Dividing Lines

You can display dividing lines between rows and column, and specify their pattern and color using [AL\\_SetDividers](#) and [AL\\_SetRGBDividers](#) commands.

### Sorting

#### Sort Buttons

User sorting of the columns via the column header sort buttons is enabled via the `userSort` parameter of [AL\\_SetSortOpts](#).

#### Sort Direction Indicator

You can use the `showSortDirIndicator` parameter of [AL\\_SetSortOpts](#) to display a sort direction indicator in the upper right corner above the vertical scroll bar. This option requires the header and the vertical scroll bar to be displayed.

When the user clicks the sort direction indicator, the sort direction of the primary sort level will be reversed and the list will be sorted. The AreaList Pro event callback (or area/form method) will run, with a \$2 event code of -1 returned to the callback method (or [AL\\_GetLastEvent](#) command (formerly [ALProEvt](#) variable), the same as if a sort button in the header was clicked (see [Using the Callback Methods](#)).

#### Sort Editor

The user can be presented with the AreaList Pro Sort Editor by calling [AL\\_ShowSortEd](#).

The window title and the prompt at the top of the window can be customized using the prompt option of [AL\\_SetSortEditorParams](#). [AL\\_SetSortEditorParams](#) also provides the ability to customize the appearance of available sort items when displaying the AreaList Pro Sort Editor.

The current sort order of the AreaList Pro area can be displayed when the Sort Editor dialog is presented by setting the `showSortOrder` parameter of [AL\\_SetSortOpts](#).

If the `allowSortEditor` option of [AL\\_SetSortOpts](#) is enabled, the user can invoke the Sort Editor by ctrl/command-clicking a column header.

[AL\\_GetSortEditorParams](#) provides the ability to retrieve the current properties of the AreaList Pro Sort Editor. If you have not previously customized the display properties, the default settings will be returned.

[AL\\_SetSortedCols](#) provides the ability to customize the default list of sorted columns.

[AL\\_GetSortedCols](#) returns the current sort columns as displayed in the Sort Editor. You should use this routine after displaying the AreaList Pro Sort Editor.

The AreaList Pro default Sort Editor is displayed as a resizable window.

## Configuring AreaList Pro Using Commands

### Procedural Sorting

Multilevel sorting can be performed procedurally on the AreaList Pro columns by using [AL\\_SetSort](#). This command will sort all of the columns in an AreaList Pro area, using up to 15 of them as sort criteria for the multi-level sort.

If a column that contains a picture column is passed as one of the sort criteria, that column and all subsequent columns will be ignored. [AL\\_GetSort](#) can be used to retrieve the current sort order of the area, regardless of whether this sort order was established by the user or procedurally.

### Sorting When Displaying Fields

Columns containing fields from a related one table will not be sorted when their column header is clicked upon.

However, if the `userSort` option of [AL\\_SetSortOpts](#) is set to 2, “Bypass the user sort buttons”, and the column header of a column containing a field from a related one table is clicked upon, the AreaList Pro event callback (or area/form method) will run, with a \$2 event code of -1 returned to the callback method (or [AL\\_GetLastEvent](#) command, formerly `ALProEvt` variable). See [Using the Callback Methods](#).

Before AreaList Pro sorts fields (using 4<sup>th</sup> Dimension’s sorting routines) it turns messages off. If messages were on previously, then AreaList Pro will turn them back on after sorting.

### Scrolling

The current scroll position can be set and retrieved using [AL\\_SetScroll](#) and [AL\\_GetScroll](#), respectively.

You can hide either the horizontal or vertical scroll bar, or both, using [AL\\_SetScroll](#).

This allows you to construct a grid of cells, providing a different interface from a standard scrolling list.

When a scroll bar is hidden, the user is still able to scroll using the Arrow keys or by dragging.

You can also set and get the scroll position procedurally.

### Selection

Use [AL\\_SetEntryOpts](#) to set the method of selection and data entry. You have extensive control over how the user interacts with a list: a mouse click can select a row, or place the cursor into the cell for data entry.

You can also configure the modifier keys (ctrl/command, shift, option/alt, and control) to control the selection behavior. Please read the section [Enterability](#).

You can configure an AreaList Pro object for no cell selection, single cell selection only, or multiple cell selection, using the `cellSelection` parameter of [AL\\_SetCellOpts](#).

If you select not to allow cell selection, then the `multiRows` parameter of [AL\\_SetRowOpts](#) is used to determine the type of row selection — single-row only or multiple rows.

## Configuring AreaList Pro Using Commands

In single-row mode, the default configuration requires that one row always be selected.

This can be overridden using the `allowNoSelection` option in `AL_SetRowOpts`, which enables the user to ctrl/command-click to deselect the selected row, leaving no rows selected. `AL_SetRowOpts` is also used to configure AreaList Pro for single or multiple rows selection mode.

You can set the selected rows using [AL\\_SetLine](#) if in single-row mode, or [AL\\_SetSelect](#) if in multiple rows selection mode.

[AL\\_GetClickedRow](#) returns the last row that was clicked, and [AL\\_GetLine](#) returns the currently selected row, as a result of a click or any other action.

You can set the selected cells using [AL\\_SetCellSel](#).

When an AreaList Pro object is in cell selection mode, mouse clicks are used to highlight cells rather than rows.

If multiple cells selection is enabled using `AL_SetCellOpts`, then the user can shift-click and ctrl/command-click to select multiple cells. Discontiguous (non-adjointing) selections are allowed.

When an AreaList Pro object is in cell selection mode, it is always possible that no cells are selected.

`AL_SetCellSel` is used to select cells procedurally, and can select a single cell, a range of cells, or a list of cells. You can determine the selected cells using [AL\\_GetCellSel](#).

When the user scrolls an AreaList Pro object that is in cell selection mode using the Arrow keys or keyboard type-ahead, the list will scroll, but the cell selection will not change.

---

***Row dragging is disabled when an AreaList Pro object is in cell selection mode.***

---

The enterability options set with `AL_SetEntryOpts` are fully supported when an AreaList Pro object is in cell selection mode.

If an AreaList Pro object is in multi-cell selection mode, the Edit menu Select All command is enabled.

## Clipboard

The data copied to the clipboard can be formatted using [AL\\_SetCopyOpts](#). This command allows you to specify the field and record delimiters copied with the data, and whether any hidden column data should be copied to the clipboard. The Edit menu Copy command is disabled when an AreaList Pro object has been set to allow cell selection using [AL\\_SetCellOpts](#).

Copying rows to the clipboard will not be allowed when displaying fields. The Copy menu item will be disabled when fields are displayed. See [Field and Record Commands](#) for more information about displaying fields.

[AL\\_SetEditMenuCallback](#) will install a callback method which will be called when any Edit menu action occurs. You have the option of overriding an 4D edit action for a given AreaList Pro area, providing an extensive customization interface when using Edit menu. See [Edit menu callbacks](#).

### Picture Columns

AreaList Pro supports the display of picture columns. The `format` parameter of [AL\\_SetFormat](#) will cause the picture to be displayed in one of several ways:

- truncated and justified to the upper left of the cell
- truncated and centered in the cell
- scaled to fit the cell
- scaled proportionally to fit the cell

The `usePicHeight` parameter of this command will tell AreaList Pro whether to use a picture's original height, which is stored with the picture, when calculating the row height for the AreaList Pro area. If you choose not to use the picture's height in the row height calculation and additional space is needed to display the picture, the `numRowLines` parameter of [AL\\_SetHeight](#) should be used to increase the row height.

### Scroll Bars — Changing Displayed Form

If an AreaList Pro object is displayed on a form in a window, and another form is going to be displayed in the window with **DIALOG**, **ADD RECORD** or **MODIFY RECORD** commands, you must inform the AreaList Pro object that another form will be displayed.

#### Overview

This is done by calling the following command whenever another form is about to be displayed:

**`AL_SetScroll`**(eList;0;0) `inform AreaList Pro object that another form will be displayed.

where `eList` is the name of the AreaList Pro object on the original form. If this is not done, the AreaList Pro object's scroll bars may be active on the other form. Scrollbars will be set back to visible next time the AreaList Pro area is redrawn, so the call above should be the last command sent to the AreaList Pro area before the new form is displayed.

**`AL_SetScroll`**(eList;0;0) hides the scrollbars until the next update of the area content. To prevent the scrollbars to appear again, use **SET VISIBLE** together with **`AL_SetScroll`**. AreaList Pro areas that are not visible will not receive any update request that may be sent to them.

#### Details: Disabling an AreaList Pro Area

The following only applies to **`AL_SetScroll`**(eList;0;0). Other parameters like -2 and -3 may hide the scrollbars, but do not have the same effect.

**`AL_SetScroll`**(eList;0;0) not only hides the scrollbars, but disables any interaction with the system (event handling) for the given area.

---

***Keep in mind that the next area update event received from 4D or the system will re-enable the area's scroll bars and event management.***

---

## Configuring AreaList Pro Using Commands

The following code will work and leave the area disabled:

```
AL_SetScroll(eList;0;0) `inform AreaList Pro object that another form will be displayed.  
DIALOG ([MyTable];"MyDialog")
```

However, the following code:

```
AL_SetScroll(eList;0;0)  
CONFIRM ("Do you want to display the dialog?")  
If(OK=1)  
  DIALOG ([MyTable];"MyDialog")  
  `do something  
End if
```

will not work, because of the **CONFIRM** dialog box, which triggers an update event to the underlying AreaList Pro area. The correct order is as follows:

```
CONFIRM ("Do you want to display the dialog?")  
If(OK=1)  
  AL_SetScroll(eList;0;0)  
  DIALOG ([MyTable];"MyDialog")  
  `do something  
End if
```

In case of doubt or complex code, the developer can also use **SET VISIBLE**:

```
SET VISIBLE (eList;False)  
AL_SetScroll (eList;0;0)  
`do something  
SET VISIBLE (eList;True)
```

If the area is not visible, it cannot get update events, therefore invisible areas disabled with **AL\_SetScroll**(eList;0;0) won't get any update event and can't be enabled until **SET VISIBLE**(eList;**True**) is called.

---

*The above is needed only when the developer displays a new form in the same window, not when the page is switched or another window is used.*

---

The **AL\_SetScroll**(area;0;0) system applies to AreaList Pro Drop Areas as well. See [Drop Area](#).



### Drag and Drop — Changing Form Pages

If the drag and drop feature of AreaList Pro is used on a multi-page layout, a similar action must be performed. When pages are changed in the layout, you must ensure that drag and drop is enabled only for AreaList Pro areas on the current page (if this feature is desired), and that drag and drop is disabled for any AreaList Pro areas on other pages. Please read the section [AreaList Pro on Multi-Page Layouts](#) for more information.

[AL\\_SetDropDst](#) should be used to disable a Drop Area on the current page when moving to a different layout page. Please read the section [Drop Area Objects on a Multi-Page Layout](#) for more information.

### Using AreaList Pro on a Resizable Window

AreaList Pro adds support for 4<sup>th</sup> Dimension resizable windows. We recommend you use 4D's built-in resizing capability. Please refer to the 4D documentation for information on making a 4D form and its objects resizable.

### Performance Issues with Formatting Commands

AreaList Pro uses an algorithm to automatically size the columns. Because of this, there is usually no need to use [AL\\_SetWidths](#) to manually size a column prior to displaying a list.

However, if the number of items in the list is very large (several thousand items with many columns), then the list might take one or two seconds to display, due to the automatic sizing calculation. If this is the case, using **AL\_SetWidths** will improve the display time of the list. Text and string columns will take the longest to automatically size. Since you can use **AL\_SetWidths** on just some of the columns, if you are displaying very large arrays, but only one is text or string, you could use the **AL\_SetWidths** command on just the text or string column, and let AreaList Pro automatically calculate the other column widths.

To determine the optimum width for a column, you can display the pixel widths of columns in the headers during your design process, and then use **AL\_SetWidths** to set the width. See [Column Widths](#) and the **AL\_SetWidths** definition for more information.

---

***When AreaList Pro displays fields, the automatic column sizing algorithm uses only the first 20 records (or less, if the selection contains less than 20 records) in the selection. These records are always read regardless of whether the columns are automatically or manually sized. Therefore there is no performance penalty using the automatic column sizing algorithm when displaying fields. See [Field and Record Commands](#) for more information about displaying fields.***

---

[AL\\_SetFormat](#) does not affect the performance of AreaList Pro, regardless of the size of the columns being displayed. This is because AreaList Pro is using 4D's array data directly, and as the list is scrolling, the formatting is being done "on-the-fly."

Sorting the columns will have the greatest impact on the time required for AreaList Pro to be displayed in the [On load](#) phase or updated in the callback method. If you will be displaying many large columns, you can reduce the display time by turning off the **automaticSort** option using [AL\\_SetSortOpts](#).



### Borders and Frames

[AL\\_SetCellBorder](#) provides the ability to set the border style for a cell.

[AL\\_SetCellFrame](#) draws a frame around a range of cells.









Both commands use RGB colors.

### Header/Cell Icon Support

#### The Escape Sentence System

AreaList Pro provides the ability to display icons in AreaList Pro headers ([AL\\_SetHeaders](#)) and cell data ([AL\\_SetFormat](#)), using picture data contained in the “cicn” or “PICT” resources, or items stored in the 4<sup>th</sup> Dimension Picture Library.

For example, when creating the arrays or header values, you can instruct AreaList Pro to display any picture type data using the following formatting options:

First Name	Last N...	Salary	Arrival	Sex	City
Mike	Erickson	\$ 1,000.00	3:10 PM 	<input type="checkbox"/>	Tapei
Don	Clark	\$ 19,822.46	1:25 AM 	<input checked="" type="checkbox"/>	Denver
Roger	Buoy	\$ 19,830.30	1:19 PM 	<input checked="" type="checkbox"/>	Telluride
John	Markoff	\$ 20,416.34	1:23 PM 	<input checked="" type="checkbox"/>	Phoenix
Russ	Havard	\$ 20,953.38	11:07 AM 	<input checked="" type="checkbox"/>	Redmond
Dan	Shafer	\$ 21,181.72	8:46 AM 	<input checked="" type="checkbox"/>	Ft. Worth
Mike	Kramer	\$ 21,337.54	6:47 AM 	<input checked="" type="checkbox"/>	San Francisco
Jim	Bartimo	\$ 21,418.88	12:59 PM 	<input checked="" type="checkbox"/>	Dallas

CELL DATA PICTURE EXAMPLE (BOOLEAN FIELD)

An “escape sentence” system can be used for headers and individual cells. If any text (cell, header, etc.) contains an escape sentence, an icon is drawn instead of the sentence. Based on the number, it may be a “cicn” resource, a “PICT” resource or a Picture Library object.

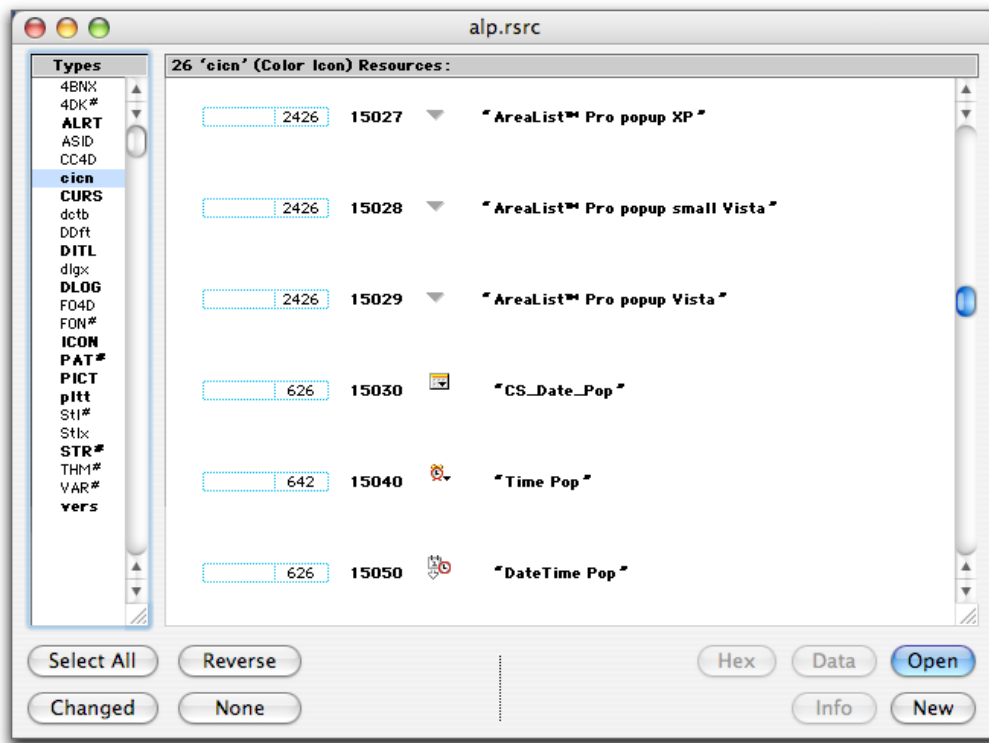
## Configuring AreaList Pro Using Commands

### Using Icons with Escape Sentences

To display an icon in the header, reference the icon resource as "**^nnnHeader**", where nnn is the desired "cicn" resource ID:

**AL\_SetHeaders**(area;1;1;"^150Header")

To display the icon at the end of the text, reference the icon resource as "**Header^nnn**" where nnn is the desired "cicn" resource ID.



CELL DATA ICON EXAMPLE (BOOLEAN FIELD)

If you want to use "PICT" resources instead of "cicn", add the 4D constant Use PICT resource to the resource ID:

**AL\_SetHeaders**(area;1;1;"^"+**String**(Use PICT resource+150) +"Header")

See the 4th Dimension Language Reference regarding the **SET LIST ITEM PROPERTIES** command, which uses the same icon syntax.

---

*When displaying icons in headers, it may be necessary to adjust the header height to accommodate the height of the icon. You can use the [AL\\_SetHeight](#) routine to increase the size of an AreaList Pro header based on your requirements.*

---

The default "escape" character (used in the call before the icon resource ID) can be modified with [AL\\_SetPictureEscape](#) and retrieved using [AL\\_GetPictureEscape](#).

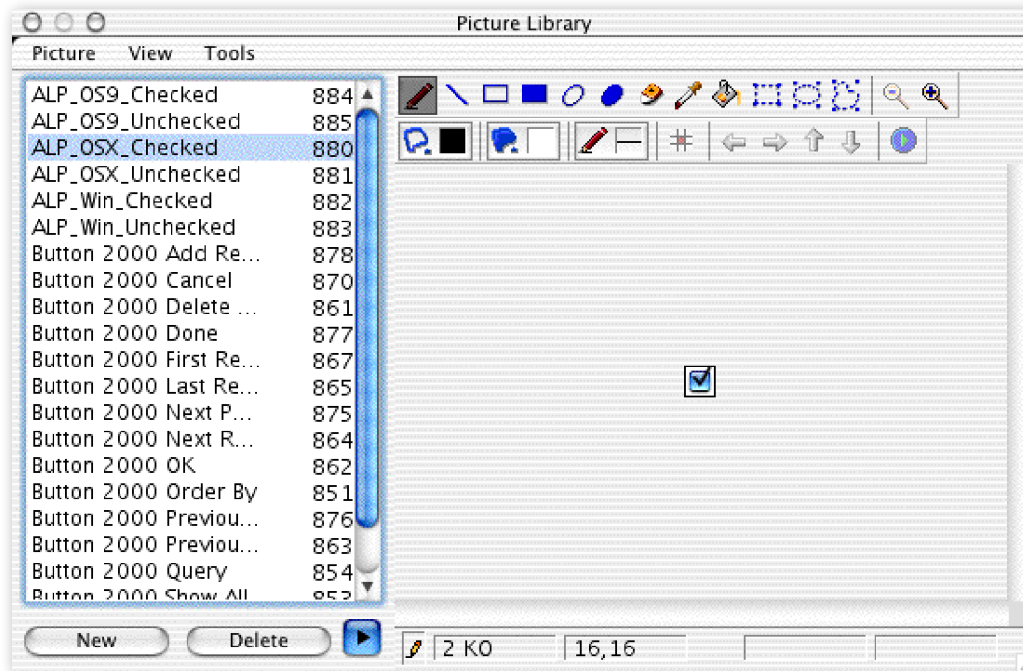
Similarly, if you wish to display icons in cell data, you would use the same technique when building the arrays for which you are using in the AreaList Pro area.

## Configuring AreaList Pro Using Commands

### Using Picture Library Items with Escape Sentences

If you would like to use an item from the 4D Picture Library, you would reference the picture ID as “Use PicRef + N”, where N is the reference number of a picture from the Design environment Picture Library.

Use PicRef is a 4D constant. See the 4th Dimension Language Reference regarding the SET LIST ITEM PROPERTIES command, which uses the same icon syntax.



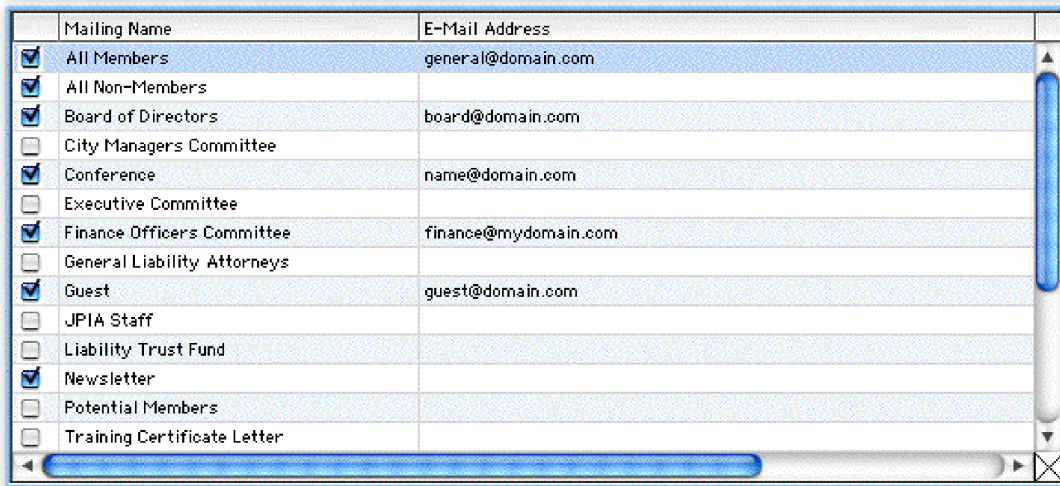
PICTURE LIBRARY CONTAINING CUSTOM CHECKBOXES

For example, if you would like to configure boolean columns to display custom checkbox icons instead of the traditional text (True;False), you can use the [AL\\_SetFormat](#) routine to provide references to icon resources contained in the 4D Picture Library.

```
$iconStr:="^"+String (Use PicRef + 880)+";"+"^"+String (Use PicRef + 881)
```

```
AL_SetFormat(eList;1;$iconStr)
```

## Configuring AreaList Pro Using Commands



	Mailing Name	E-Mail Address
<input checked="" type="checkbox"/>	All Members	general@domain.com
<input checked="" type="checkbox"/>	All Non-Members	
<input checked="" type="checkbox"/>	Board of Directors	board@domain.com
<input type="checkbox"/>	City Managers Committee	
<input checked="" type="checkbox"/>	Conference	name@domain.com
<input type="checkbox"/>	Executive Committee	
<input checked="" type="checkbox"/>	Finance Officers Committee	finance@mydomain.com
<input type="checkbox"/>	General Liability Attorneys	
<input checked="" type="checkbox"/>	Guest	guest@domain.com
<input type="checkbox"/>	JPIA Staff	
<input type="checkbox"/>	Liability Trust Fund	
<input checked="" type="checkbox"/>	Newsletter	
<input type="checkbox"/>	Potential Members	
<input type="checkbox"/>	Training Certificate Letter	

CUSTOMIZED BOOLEAN COLUMN USING CHECKBOX ICONS FROM 4D PICTURE LIBRARY

The default “escape” character (used in the call before the icon Picture Library ID) can be modified with [AL\\_SetPictureEscape](#) and retrieved using [AL\\_GetPictureEscape](#).

## Longint Reference System

Resources and Picture Library items are also used by [AL\\_SetCellIcon](#), which places icons into individual cells and [AL\\_SetHeaderOptions](#), which provides the ability to customize the interface over the scrollbars (sort area). [AL\\_GetHeaderOptions](#) returns the current setting for the area.

These routines include an **iconRef** parameter, which is one of the following:

- N, where N is the resource ID of Mac OS-based “cicn” resource
- [Use PICT resource](#) + N, where N is the the resource ID of a Mac OS-based “PICT” resource
- [Use PicRef](#) + N, where N is the reference number of a picture from the Design environment Picture Library
- pass zero (0) if you do not want any icon for the header or the cell

## Picture Objects in Headers

In addition, [AL\\_SetHeaderIcon](#) provides the ability to procedurally place icons in column headers using 4D picture objects (fields or variables).

# Commands

## AL\_Register

(registrationKey:S) → resultCode:L

Parameter	Type	Description
→ registrationKey	string	Registration key
← resultCode	longint	Result code

**AL\_Register** is used to register the AreaList Pro plug-in for standalone or server use. You must call **AL\_Register** with a valid registration key; otherwise AreaList Pro will operate in demonstration mode.

Without a valid registration key, AreaList Pro will operate in demonstration mode during 20 minutes.

Like all e-Node plug-ins, AreaList Pro offers six different license types. There are no such things as MacOS vs Windows or Development vs Deployment:

- **Single user license.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Standalone or Runtime. Since the registration key is linked to a specific 4D license, you need to provide the number returned by the 4D command **GET SERIAL INFORMATION** (first parameter). A new license will be supplied for free at any time if you change your 4D version and/or get a new 4D registration key, provided that your previous licenses match the current public version at the exchange time.
- **Small server.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Server up to 10 users. The registration key is linked to your 4D Server license just as above.
- **Medium server.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Server up with 11 to 20 users. The registration key is linked to your 4D Server license just as above.
- **Large server.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on 4D Server over 20 users. The registration key is linked to your 4D Server license just as above.
- **Unlimited Single User.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on as many 4D Standalone, Runtime or Engine copies that run your 4D application(s). This is a yearly license, which expires after the date when it is to be renewed. The expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire.**
- **Unlimited OEM.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode) on as many 4D Server (of any number of users), 4D Standalone, Runtime or Engine copies that run your 4D application(s). This is a yearly license, which expires after the date when it is to be renewed. The expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire.**

A 4D database used to retrieve your 4D serial information is available from the following link:

<http://www.e-node.net/ftp/GetSerialInfo>

## Configuring AreaList Pro Using Commands

---

*The registration system has been modified in version 8.3. Only one registration key is now required.*

---

**registrationKey** — Pass the registration key to register your copy of AreaList Pro. Only one registration key is required. The key is either linked to the 4D or 4D Server serial number (individual licenses), or to the name of the company/developer (unlimited annual licenses).

---

*Multiple calls to **AL\_Register** are allowed. The plug-in will be activated if at least one valid key is used.*

---

**resultCode** — This will return a value of 1 if the registration key is valid and a value of 0 if the registration key is invalid. You should verify the correctness of the registration key by tracing over the call to **AL\_Register** and examining **resultCode**.

Example:

```
C_LONGINT($result)
$result:=AL_Register("Place your registration key here")
If($result#1) `error
    ALERT("AreaList Pro could not be registered:"+String($result))
End if
```

Example with multiple calls:

```
C_LONGINT($result) `ignored in this case
$result:=AL_Register("Registration key one")
$result:=AL_Register("Registration key two")
$result:=AL_Register("Registration key three")
`etc.
```

---

## %AreaListPro

**%AreaListPro** is the command used to identify the AreaList Pro plug-in area when you create a plug-in area object on a layout. This command is only used in the object definition for an AreaList Pro object, and should never be used as a command in a method.



### AL\_SetArraysNam

(areaRef:L; columnNumber:I; numArrays:I; array1:S; ...; arrayN:S) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to set the first array
→ numArrays	integer	Number of arrays to set (up to 15)
→ array1; ...; arrayN	string	Names of 4D arrays
← resultCode	integer	Result code

**AL\_SetArraysNam** tells AreaList Pro what arrays to display. Up to fifteen arrays can be set at a time. Any 4D array type can be used except pointer and two-dimensional arrays.

There are three very important points to note about this command:

- This command must be called first, before any of the other commands, in the On load phase or in another phase (form event).
- The columns must be added in sequential order, unless the particular column has already been added. In other words, to set 30 arrays, you must set arrays 1 through 15 prior to setting arrays 16 through 30.
- All arrays set with this command must have the same number of elements as each other and as any other arrays previously set.

**AL\_SetArraysNam** may be called in the On load phase to initially set the arrays to be displayed. Since AreaList Pro can display up to 512 arrays, this command may have to be used more than once. However, it is not mandatory to set any arrays in the On load phase; in that case the area on the layout where AreaList Pro is defined will be blank.

**AL\_SetArraysNam** may be called in other phases (form events) to set arrays to be displayed or to replace arrays that are already displayed.

You can pass process arrays and interprocess arrays to AreaList Pro, but not local arrays (a local array has a name that starts with a "\$" character; an interprocess array has a name that starts with a "Q" character on MacOS and the "<>" characters on Windows).

One dimension of a two-dimensional array may be passed in the **array1; ...; arrayN** parameters. For example: "my2DArray{1}" may be passed as **array1**.

**areaRef** — AreaList Pro area reference.

**columnNumber** — This parameter specifies the column number to set the first array being passed by this call of **AL\_SetArraysNam**.

**numArrays** — This parameter specifies the number of columns being set with this call to **AL\_SetArraysNam**.

## Configuring AreaList Pro Using Commands

**resultCode** — The possible values are:

Constant	Value	Action
AL No error in arrays	0	
AL Not an array error	1	Check to make sure all arrays are correctly typed
AL Wrong type array error	2	Pointer and two-dimensional arrays are not allowed
AL Wrong number rows error	3	Make sure that all arrays have the same number of elements
AL Max arrays exceeded error	4	512 arrays is the maximum
AL Low memory array error	5	Increase 4D's RAM partition, or change your approach to use fewer or smaller arrays

Examples:

### Case of

**:(Form event=On Load)**

**SELECTION TO ARRAY** ([Contacts]FN;aFN;[Contacts]LN;aLN;[Contacts]City;aCity;[Contacts]State;aState) `load the arrays

\$error:=**AL\_SetArraysNam**(eList;1;4;"aFN";"aLN";"aCity";"aState") `starting at column 1, set 4 arrays

### End case

`Set up the eList AreaList Pro object with 25 arrays

`two calls must be made since only 15 arrays can be passed each time

\$error:=**AL\_SetArraysNam**(eList;1;15;"array1";"array2";"array3";"array4";"array5";"array6";"array7";"array8";"array9";"array10";"array11";"array12";"array13";"array14";"array15")

\$error:=**AL\_SetArraysNam**(eList;16;10;"array16";"array17";"array18";"array19";"array20";"array21";"array22";"array23";"array24";"array25")

---

## AL\_InsArrayNam

(areaRef:L; columnNumber:I; numArrays:I; array1:S; ...; arrayN:S) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to set the first array
→ numArrays	integer	Number of arrays to set (up to 15)
→ array1; ...; arrayN	string	Names of 4D arrays
← resultCode	integer	Result code

**AL\_InsArrayNam** functions the same as **AL\_SetArraysNam**, except that the arrays are inserted before columnNumber.



## Configuring AreaList Pro Using Commands

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding array.

Up to fifteen arrays can be set at a time. Any 4D array type can be used except pointer and two-dimensional arrays. There are three very important points to note about this command:

- This command (or **AL\_SetArraysNam**) must be called first, before any of the other commands, in the On load phase or in another phase (form event).
- The columns must be added in sequential order, unless the particular column has already been added. In other words, to set 30 arrays, you must set arrays 1 through 15 prior to setting arrays 16 through 30.
- All arrays set with this command must have the same number of elements as each other and as any other arrays previously set.

**AL\_InsArrayNam** may be called in the On load phase to initially set the arrays to be displayed. Since AreaList Pro can display up to 512 arrays, this command may have to be used more than once. However, it is not mandatory to set any arrays in the On load phase; in that case the area on the layout where AreaList Pro is defined will be blank.

You can pass process arrays and interprocess arrays to AreaList Pro, but not local arrays (a local array has a name that starts with a "\$" character; an interprocess array has a name that starts with a "◇" character on MacOS and the "<>" characters on Windows).

One dimension of a two-dimensional array may be passed in the **array1; ...; arrayN** parameters. For example: "my2DArray{1}" may be passed as **array1**.

**areaRef** — AreaList Pro area reference.

**columnNumber** — This parameter specifies the column number to insert the first array being passed by this call of **AL\_InsArrayNam**.

**numArrays** — This parameter specifies the number of columns being set with this call to **AL\_InsArrayNam**.

**resultCode** — The possible values are:

Constant	Value	Action
AL No error in arrays	0	
AL Not an array error	1	Check to make sure all arrays are correctly typed
AL Wrong type array error	2	Pointer and two-dimensional arrays are not allowed
AL Wrong number rows error	3	Make sure that all arrays have the same number of elements
AL Max arrays exceeded error	4	512 arrays is the maximum
AL Low memory array error	5	Increase 4D's RAM partition, or change your approach to use fewer or smaller arrays

Example:

```
$error:=AL_InsArrayNam(eList;4;3;"aFN";"aLN";"aComp") `starting at column 4, insert 3 arrays
```

## Configuring AreaList Pro Using Commands

### AL\_GetArrayNames

(areaRef:L; resultArray:X; options:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ resultArray	array (string or text)	Result array of array names
→ options	longint	Return options
← resultCode	longint	Result code

**AL\_GetArrayNames** will return an array of array names (when using array display — see [AL\\_GetMode](#)). If you have used [AL\\_InsArrayNam](#), this routine will not work. You can only use this routine when configuring arrays via [AL\\_SetArraysNam](#).

When using this routine, you may return either the complete list of arrays used to setup the list, or only the visible arrays (see **options** parameter).

**areaRef** — AreaList Pro area reference.

**resultArray** — A valid 4<sup>th</sup> Dimension array (by reference) which will receive the list of array names.

**options** — Return options:

0 — return only visible arrays (default)

1 — return all arrays

**resultCode** — The possible values are:

Value	Result Code	Action
0	No error	
-50	Parameter error	Array of wrong type — must be string or text array
-1	Wrong mode	See <a href="#">AL_GetMode</a>
-2	Arrays were not created using <b>AL_SetArraysNam</b>	Make sure that all arrays have been created using <b>AL_SetArraysNam</b> , not <b>AL_InsertArrays</b>

### AL\_RemoveArrays

(areaRef:L; columnNumber:I; numArrays:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to remove the first array
→ numArrays	integer	Number of arrays to remove (up to 512)

**AL\_RemoveArrays** is used to remove arrays from AreaList Pro. **numArrays**, beginning at **columnNumber**, will be removed from the list.

## Configuring AreaList Pro Using Commands

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding array.

Examples:

**AL\_RemoveArrays**(eList;8;4) `starting at column 8, remove 4 arrays

**AL\_RemoveArrays**(eList;1;20) `remove all 20 arrays

---

### AL\_UpdateArrays

(areaRef:L; updateMethod:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ updateMethod	integer	Method to use to update the AreaList Pro object

**AL\_UpdateArrays** is used to update AreaList Pro. Use this command whenever any elements of the arrays being displayed are changed (elements added, deleted, or modified), but the arrays themselves remain the same.

---

**Warning:** *in an enterable area, the row containing the currently edited data must not be deleted. [AL\\_ExitCell](#) must be called before the row (array element) is deleted.*

---

**AL\_UpdateArrays** must be called after modifying the arrays and before any other setup commands (sorting, formatting, etc.).

**updateMethod** — This parameter tells AreaList Pro how to update the AreaList Pro object **areaRef**.

Constant	Value	Description	When to Use
AL Recalculate arrays	-2	Rescan all arrays and recalculate all applicable heights, widths, and other related values. The scroll position, and row or cell selection will be reset.	If column or row resizing is necessary, or you have added or removed elements to any of the displayed arrays. Also if you show or hide either scroll bar, the headers, or footers, or add or remove arrays.
AL Refresh and update arrays	-1	Refresh the AreaList Pro object, but don't recalculate any values.	The AreaList Pro object needs to be updated because of changes to an array element's contents, or formatting changes to colors, styles, etc. This value should only be used when no column or row resizing is necessary, since formatting, styles, or an element's new contents could affect a column width or row height.

---

*You may only pass a value of -1 for updateMethod when calling AL\_UpdateArrays from a callback method **other than the event callback**. -2 can be used in event callback methods.*

*Please read the section [Using Callback Methods During Data Entry](#) for more information.*

---

## Configuring AreaList Pro Using Commands

Examples:

```
`Any action which modifies an array element value, or changes a configuration attribute
`must include updating the AreaList Pro object
AL_UpdateArrays(eList;-2)

`bDeleteRows button method
`This example shows how to delete elements from displayed arrays and how to update AreaList Pro
`The routine deletes selected rows in an AreaList Pro object named eList
`eList is configured for multiple rows selection, and it is displaying three arrays: aFN, aLN, aComp
ARRAY LONGINT(aRows;0) `create an long integer array with a size of zero
$OK:=AL_GetSelect(eList;aRows) `get the rows selected by the user, put into aRows array
If($OK=1) `enough RAM was available to resize the aRows array
  For($i;Size of array(aRows);1;-1) `start at the end of the array and go to top
    DELETE ELEMENT(aFN;aRows{$i}) `delete the selected element from the three arrays
    DELETE ELEMENT(aLN;aRows{$i})
    DELETE ELEMENT(aComp;aRows{$i})
  End for
  AL_GetScroll(eList;vVert;vHoriz) `get current scroll position
  AL_UpdateArrays(eList;-2) `update the AreaList Pro object
  AL_SetScroll(eList;vVert;vHoriz) `reset scroll position so it doesn't change
End if
```

---

## AL\_SetHeaders

(areaRef:L; columnNumber:I; numHeaders:I; header1:S; ...; headerN:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to set up the first header
→ numHeaders	integer	Number of headers to set (up to 15)
→ header1; ...; headerN	string	Values to display in column headers

**AL\_SetHeaders** is used to specify the value to display in the header for each column. Up to fifteen headers can be set at a time.

The size of the header value is used by the automatic column sizing algorithm. If you are displaying a fixed-string array with an element size of 2 characters, the column will be very narrow, unless you specify a header which contains several characters.

For example, states are usually stored in a database as a two-character alpha, and you would probably display them directly or load them into a string array sized for two-characters length. But if you specify a header of "State" the column will be sized about two and a half times wider.

## Configuring AreaList Pro Using Commands

If the header length is less than the values being displayed in the column, then the header length will not affect the column width.

A, B, C, etc. will be displayed in the headers if **AL\_SetHeaders** is not used. The **AL\_SetHeaders** command can be used in the On load phase or in another phase (form event).

Examples:

```
$error:=AL_SetArraysNam(eList;1;4;"aFN";"aLN";"aCity";"aState")  
AL_SetHeaders(eList;1;4;"First Name";"Last Name";"City";"State")
```

```
$error:=AL_SetArraysNam(eList;1;2;"aFN";"aLN")  
AL_SetHeaders(eList;1;2;Field name ([[People]FirstName]);Field name ([[People]LastName]))
```

AreaList Pro provides the ability to display icons in AreaList Pro headers. See [Header/Cell Icon Support](#) for information about the use of **AL\_SetHeaders** to display icons in column headers, using picture data contained in the "cicn" or "PICT" resources, or items stored in the 4<sup>th</sup> Dimension Picture Library.

AreaList Pro also includes modern column headers, including direct platform detection. See [Headers](#).

Vista specific behavior: if AreaList Pro does not have enough space (at least 13 pixels) to display the header, it will be automatically resized to fit. In addition, when using AreaList Pro with Vista, the sort arrow is centered above the column header, thus you need to take care to make sure that the header is high enough.

---

## AL\_GetHeaders

(areaRef:L; headerList:X; options:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← headerList	array (text or string)	Result header list
→ options	longint	Return options
← resultCode	longint	Result code

**AL\_GetHeaders** will return an array of all headers for the defined AreaList Pro area. You may optionally return only visible headers using the **options** parameter.

**headerList** — A valid 4<sup>th</sup> Dimension array (text or string) which will contain a list of all area headers.

**options** — When extracting the list of headers names, you may optionally return only visible headers:

0 — no options, returns all headers (default)

1 — returns only visible headers

**resultCode** — Returns an error code, or 0 is no error occurred.

The following will build an AreaList Pro area based on field references from a parent and related table.

## Configuring AreaList Pro Using Commands

```
AL_SetHeaders(eList;1;1;"First Name")
```

```
AL_SetHeaders(eList;2;1;"Last Name")
```

Then, we'll use the **AL\_GetHeaders** routine to extract the header names.

```
ARRAY TEXT (atAL_HeaderList;0)
```

```
$ret:=AL_GetHeaders(eList;atAL_HeaderList)
```

When the routine is complete, the array will have two elements:

```
atAL_HeaderList{1} contains "First Name"
```

```
atAL_HeaderList{2} contains "Last Name"
```

---

## AL\_SetHeaderIcon

(areaRef:L; columnNumber:I; iconAlignment:I picture:P; horPosition:I; vertPosition:I; offset:I; scaling:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to set the header icon
→ iconAlignment	integer	Position of icon
→ picture	picture	Icon or picture to use
→ horPosition	integer	Horizontal position
→ vertPosition	integer	Vertical position
→ offset	integer	Pixel offset
→ scaling	integer	Scaling

**AL\_SetHeaderIcon** provides the ability to procedurally place icons in column headers. One or two icons may be used (left and right).

**columnNumber** — Desired header column number.

**iconAlignment** — Position of icon (a header can contain up to two icons):

**0** — places icon on left of header

**1** — places icon on right of header

**picture** — 4D picture object containing the icon (due to limitations of icons drawing in headers, you must first load the desired icon into a 4D picture object).

**horPosition** — One the following options:

**0** — default (left for left icon, right for right icon)

**1** — align left

**2** — align center

**3** — align right

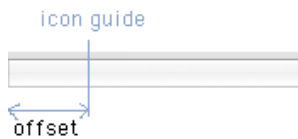
## Configuring AreaList Pro Using Commands

**vertPosition** — One the following options:

- 0** — default (top)
- 1** — align top
- 2** — align center
- 3** — align bottom

**offset** — Offset of the “icon guide”. The horizontal position is relative to this position. If the horizontal alignment is center, the icon is centered between the guide and corresponding side of cell (left for left icon, right for right icon).

The picture below illustrates the icon guide and its offset:



ICON GUIDE AND OFFSET

In the picture below, the left icon is aligned right to the icon guide and the right icon is aligned left to the icon guide:



LEFT ICON ALIGNED RIGHT - RIGHT ICON ALIGNED LEFT

In the picture below, the left icon is centered between the left border and the icon guide and no right icon is used:



LEFT ICON CENTERED

**scaling** — One the following options:

- 0** — truncated
- 1** — scaled

The cell content (text) is drawn into the space that is left once the icon is drawn. If the icon is larger than the remaining available space, the text is drawn over the icon.

For example, if the column width is 100 pixels and you draw a 15 pixel icon, there is remaining width of 85 pixels where the text will be drawn. If, however, the total width (icon + text) exceeds the column width, the text will be drawn over the picture. This allows background pictures behind the text.

## Configuring AreaList Pro Using Commands

The following example will use the same icon as [AL\\_SetCellIcon](#), but it will first load the icon into a 4D picture object:

```
C_PICTURE($pict)
C_INTEGER($col;$iconAlign;$horPos;$verPos;$offset;$scaling)
$col:=3 `place icon in 3rd column
$iconAlign:=0 `draw on left
$horPos:=0 `default
$verPos:=2 `align center
$offset:=5
$scaling:=0
GET PICTURE FROM LIBRARY(1717;$pict)
AL_SetHeaderIcon(eAL_Output;$col;$iconAlign;$pict;$horPos;$verPos;$offset;$scaling)
```

---

## AL\_SetHeaderOptions

(areaRef:L; options:L; iconRef:L; callbackMethod:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ options	longint	Options attribute
→ iconRef	longint	Reference of the icon or picture to use
→ callbackMethod	string	Header callback method

**AL\_SetHeaderOptions** provides the ability to customize the interface over the scrollbars (sort area). You can customize the icon which is displayed using a “cicn” or “PICT” resource, or an item from the 4D Picture Library (see details below).

---

***For optimal results, the icon size should be 13w x 12h.***

---

**options** — Desired options for overriding the sort icon:

- 0** — no options, use default interface
- 1** — display custom icon and execute **callbackMethod** on mouseUp
- 2** — display custom icon and execute **callbackMethod** on mouseDown

**iconRef** — Reference of the icon or picture to use. Both “cicn” and “PICT” resources can be used, as well as items from the Picture Library.



## Configuring AreaList Pro Using Commands

To associate an icon to the header, pass one of the following numeric values in **iconRef** (Use PICT resource and Use PicRef are 4D constants):

- N, where N is the resource ID of Mac OS-based “cicn” resource
- Use PICT resource + N, where N is the the resource ID of a Mac OS-based “PICT” resource
- Use PicRef + N, where N is the reference number of a picture from the Design environment Picture Library
- pass zero (0) if you do not want any icon for the header

See [Header/Cell Icon Support](#) for examples. See also the 4<sup>th</sup> Dimension Language Reference regarding the **SET LIST ITEM PROPERTIES** command, which uses the same icon syntax.

**callbackMethod** — Desired callback method, which is executed when the icon is clicked:

- if you have passed an option value of 1, the callback will be executed when the user releases the mouse button
- if you have passed an option value of 2, the callback will be executed immediately when user clicks the icon

The icon clicked callback method is passed one parameter by AreaList Pro. This parameter is a long integer that corresponds to the name of the AreaList Pro object on the layout.

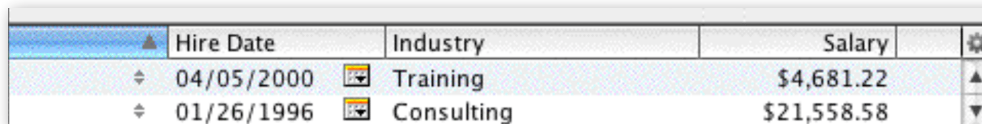
You must use the following declaration in your callback method:

**C\_LONGINT** (\$1)

Since the long integer \$1 parameter contains 4D’s representation of the AreaList Pro object, it can be used as the first parameter of any AreaList Pro method called.

The following example will create a custom icon, using a 4D Picture Library item (ID = \$pictLibID):

**AL\_SetHeaderOptions**(\$AL\_AREA;2;\$pictLibID+Use PicRef;"SortIconCallback")



Hire Date	Industry	Salary
04/05/2000	Training	\$4,681.22
01/26/1996	Consulting	\$21,558.58

CUSTOM HEADER ICON AND ACTION

## AL\_GetHeaderOptions

(areaRef:L; options:L; iconRef:L; callbackMethod:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← options	longint	Options attribute
← iconRef	longint	Reference of the icon or picture
← callbackMethod	string	Header callback method

**AL\_GetHeaderOptions** will return the attributes set by [AL\\_SetHeaderOptions](#).

### AL\_SetFooters

(areaRef:L; columnNumber:I; numFooters:I; footer1:S; ...; footerN:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to set the first footer
→ numFooters	integer	Number of footers to set (up to 15)
→ footer1; ...; footerN	string	Values to display in column footers

**AL\_SetFooters** is used to specify the value to display in the footer for each column. Up to fifteen footers can be set at a time. The **showFooters** option of [AL\\_SetMiscOpts](#) must be enabled.

The size of the footer value is used by the automatic column sizing algorithm the same way that the header for a column is used. For more information, see [AL\\_SetHeaders](#).

Nothing will be displayed in the footer area if **AL\_SetFooters** is not used. **AL\_SetFooters** can be used in the On load phase or in another phase (form event).

Example:

```
For ($i;1;Size of array (aSalary))
    $Total:= $Total+aSalary{$i}
End for
AL_SetFooters(eEmplList; 3;1;String($Total))
```

### AL\_GetFooters

(areaRef:L; footerList:X; options:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← footerList	array (text or string)	Footer value list
→ options	longint	Option reference
← resultCode	longint	Result code

**AL\_GetFooters** will return the footer information if you have enabled footers (see [AL\\_SetColOpts](#)). When calling **AL\_GetFooters**, you have the option of including or omitting invisible column(s).

**footerList** — A valid 4<sup>th</sup> Dimension array (text or string) which will contain a list of all area footers.

## Configuring AreaList Pro Using Commands

**options** — When extracting the list of footers names, you may optionally return only visible column footers:

**0** — no options, returns all footers (default)

**1** — returns only visible column footers

The following example will return all the footer values from visible columns only:

```
AL_SetMiscOpts(eList;0;0;"";1;1)
```

```
AL_SetColOpts(eList;1;0;0;1) `hide last column
```

```
AL_SetFooters(eList;1;1;"Footer1")
```

```
AL_SetFooters(eList;2;1;"Footer2")
```

```
AL_SetFooters(eList;3;1;"Footer3")
```

```
ARRAY TEXT(atAL_FooterList;0)
```

```
$ret:=AL_GetFooters(eList;atAL_FooterList;1) `retrieve footer data, only visible columns
```

When the routine is complete, **atAL\_FooterList** will contain two elements:

atAL\_FooterList{1} contains "Footer1"

atAL\_FooterList{2} contains "Footer2"

---

## AL\_SetWidths

(areaRef:L; columnNumber:I; numWidths:I; width1:I; ...; widthN:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to set the first width
→ numWidths	integer	Number of widths to set (up to 15)
→ width1; ...; widthN	integer	Pixel widths of columns

**AL\_SetWidths** is used to set the pixel width for one or more columns. Up to fifteen widths can be set at a time. A width of zero forces a column to be sized automatically based on its data type.

A column cannot be less than 3 pixels wide. If you pass a value of less than 3 but greater than zero, AreaList Pro will ignore it and use 3. AreaList Pro will not let a column be wider than the width of the list area minus 20.

If not called, the default width for all columns is determined based on the type of array or field displayed in the column and the footer for the column.

**AL\_SetWidths** can be used in the On load phase or in another phase (form event).

Example:

```
$error:=AL_SetArraysNam(eList;1;5;"aFN";"aLN";"aCity";"aState";"aZip")
```

```
AL_SetWidths(eList;1;5;150;50;0;100;0) `0 forces autosizing for that column
```

You can get the column widths using [AL\\_GetWidths](#).

### AL\_SetFormat

(areaRef:L; columnNumber:I; format:S; columnJust:I; headerJust:I; footerJust:I; usePictHeight:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to set the format and justification
→ format	string	Format to use
→ columnJust	integer	Justification for column list items
→ headerJust	integer	Justification for column header
→ footerJust	integer	Justification for column footer
→ usePictHeight	integer	Use the picture height in the row height calculation

**AL\_SetFormat** is used to control the format and justification of a column being displayed. You can control the format of string, integer, long integer, real, date, boolean, and picture columns with the **format** parameter. Time values can be formatted also, since they use long integer arrays. Any valid 4D format, including custom formats created in the Design environment, may be used with these column types, except for string arrays. Text columns cannot be formatted.

Additionally, null time and date values can be set to display a blank by appending a dash character ("-") to the **format** string parameter.

The defaults for the different column types are:

Column Type	Format
Integer	"##,##0"
Long Integer	"#,###,##0"
Real	"#,###,##0.00"
Boolean	"True;False"
Date	"0"
Picture	"0"

These values are initialized at startup from the STR# 15023 resource.

In European versions, this resource has been modified as follows:

Column Type	Format
Integer	"## ##0"
Long Integer	"# ### ##0"
Real	"# ### ##0,00"

In addition, the French version of AreaList Pro initializes the boolean format as "Vrai;Faux".

See [AL\\_SetDefaultFormat](#), which can be used to modify the default formats for all AreaList Pro areas.

## Configuring AreaList Pro Using Commands

**format** (for string arrays) — Any formatting characters supported for 4D are allowed. Pre-defined styles (i.e. those saved in the Design environment) are not allowed.

**format** (for text arrays) — Not supported.

**format** (for numeric arrays) — See the 4D command **String** in the 4D Language Reference for the possible values. Any valid 4D numeric format may be used.

**format** (for boolean arrays) — The string contains two formats, one for the **True** value, the other for the **False** value, separated by a semicolon. Examples: "Male;Female" and "MacOS;Windows."

**format** (for date arrays) — See the 4D **String** command in the 4D Language Reference for the possible values. Any valid 4D date format may be used. Examples: "0" or "3" are valid formats.

Format	Example
0	09/20/07 (default)
1	9/20/07
2	Thu, Sep 20, 2007
3	Thursday, September 20, 2007
4	09/20/07 or 09/20/1997
5	September 20, 2007
6	Sep 20, 2007

**format** (for "time" arrays) — See the 4D **String** command in the 4D Language Reference, and the 4D Design Reference discussion of formatting for the possible values. There are no time arrays in 4D as such, they are in reality long integer arrays. These arrays are displayed as time **AL\_SetFormat** values by using the proper format. The format is the two character sequence "&/" followed by the number given in the discussion of the **String** command. For example, one proper format for a time array would be "&/2".

Format	Example
1	01:02:03
2	01:02
3	1 hour 2 minutes 3 seconds
4	1 hour 2 minutes
5	1:02 AM

**format** (for picture arrays):

- 0** — the picture will be truncated, if necessary, and justified to the upper left (default)
- 1** — the picture will be truncated, if necessary, and centered in the cell
- 2** — the picture will be scaled to fit the cell
- 3** — the picture will be scaled to fit the cell, and remain proportional to its original size

## Configuring AreaList Pro Using Commands

**columnJust**, **headerJust**, and **footerJust** — The justification for a column, its header, and its footer can be controlled independently. The possible values are:

Value	Justification
0	Default
1	Left
2	Center
3	Right

By default, headers are left justified, unless the column elements are center justified. In that case, the header will default to center justification.

The default footer justification corresponds to the column justifications, which for the different column types are:

Column Type	Default Column Justification
Integer	right
Long Integer (including Time)	right
Real	right
Boolean	left
Date	right
String	left
Text	left
Picture	n/a — see the <b>format</b> parameter

The **columnJust** parameter is ignored for picture columns. Use the **format** parameter to justify picture columns.

**usePictHeight**:

- 0** — ignore the picture height when calculating the row height (default)
- 1** — use height of the largest picture when calculating the row height

If the column **columnNumber** does not have a picture column, this parameter will be ignored.

**AL\_SetFormat** can be used in the On load phase or in another phase (form event).

Examples:

`Format a real column (3rd column), default column justification, center header justification, and default footer justification

**AL\_SetFormat**(names;3;"\$###,###.00";0;2;0;0)

`Format a string (2nd column), default column justification and default header justification, center footer justification

**AL\_SetFormat**(eContacts;2;"(###) ###-####";0;0;2;0)

`Format a boolean column (4th column), right column justification and left header justification

**AL\_SetFormat**(eList;4;"Male;Female";3;1;0;0)

`Format style 3 for a date column, default justification (5th column), default column, header, and footer justification

**AL\_SetFormat**(eList;5;"3")

## Configuring AreaList Pro Using Commands

`Format style 2 for a time column, right justification for header and column (7th column)

**AL\_SetFormat**(eList;7;"&/2";3;3;0;0)

`Custom format style, default justification for column, center header (5th column)

**AL\_SetFormat**(eList;5;"I Dollars";0;2;0;0)

`Scale picture column to fit proportionally (1st column), use default header and footer justification, and use picture size in row height calculation

**AL\_SetFormat**(eList;1;"3";0;0;0;1)

See also [Header/Cell Icon Support](#) for information about the use of **AL\_SetFormat** to display icons in cell data, using picture data contained in the "cicn" or "PICT" resources, or items stored in the 4<sup>th</sup> Dimension Picture Library.

## AL\_SetDefaultFormat

(selector:L; format:S)

Parameter	Type	Description
→ selector	longint	Selector (data type for which to modify the default)
→ format	string	Desired format

**AL\_SetDefaultFormat** sets the default **format** for the specified type to apply to all AreaList Pro areas to be created. Existing areas are not affected.

The **format** can eventually be modified for any column of any area using [AL\\_SetFormat](#).

**selector** — indicates the data type for which to set the default format:

Constant	Value	Data Type
AL Format Integer	1	Integer
AL Format Longint	2	Long Integer
AL Format Real	3	Real
AL Format Boolean	4	Boolean
AL Format Date	5	Date
AL Format Picture	6	Picture

**format** — Format to use for the data type specified by **selector**. The **format** is specified as a string. See [AL\\_SetFormat](#).

Example:

`Modify the default format for pictures ("0" = truncated, if necessary, and justified to the upper left)  
to "2" = scaled to fit the cell

**AL\_SetDefaultFormat**(AL Format Picture;"2")



### AL\_GetFormat

(areaRef:L; columnNumber:I; format:S; columnJust:I; headerJust:I; footerJust:I; usePictHeight:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	longint	Desired list column
← format	string	Format used
← columnJust	integer	Justification for column list items
← headerJust	integer	Justification for column header
← footerJust	integer	Justification for column footer
← usePictHeight	integer	Use the picture height in the row height calculation

**AL\_GetFormat** will return the formatting attributes for the supplied column number (see [AL\\_SetFormat](#) for information on setting column display attributes).

**format** — Returns the defined column format string. There are many formatting options, which can be used to customize the appearance of a given AreaList Pro column. For a complete list of formatting options, please refer to the **AL\_SetFormat** routine.

**columnJust** — Returns the defined column justification value.

**headerJust** — Returns the defined header justification value.

**footerJust** — Returns the defined footer justification value.

**usePictHeight** — Returns the **usePictHeight** property.

The following example will set the formatting attributes for a given AreaList Pro area cell:

```
`Format a real column (3rd column), default column justification, center header justification,
  and default footer justification
AL_SetFormat(eList;3;"$###,###.00";0;2;0;0)
```

Then, we'll use the **AL\_GetFormat** routine to extract the formatting values:

```
C_STRING (32;$sFormat)
C_LONGINT ($colJust;$headerJust;$footerJust;$usePictHeight)
AL_GetFormat(eList;3;$sFormat;$colJust;$headerJust;$footerJust;$usePictHeight)
```

When the routine has completed, the following values will be returned:

```
$sFormat contains "$###,###.00"
$colJust contains 0
$headerJust contains 2
$footerJust contains 0
$usePictHeight contains 0
```

### AL\_SetHdrStyle

(areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column for which to set the header style
→ fontName	string	Name of the font to use
→ size	integer	Size of the font
→ styleNum	integer	Style of the font

**AL\_SetHdrStyle** is used to control the appearance of the AreaList Pro column headers. The columns can be controlled individually or as a group.

**columnNumber** — This parameter specifies what column header to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

**fontName** — Use this parameter to specify the font for the specified **columnNumber**. If not called, or the specified **fontName** is not found, the header(s) will be displayed with the default font. See [AL\\_SetDefaultStyle](#). If the font specified by **fontName** is not installed, then the default font will be used.

**size** — Use this parameter to specify the font size for the specified **columnNumber**. If not called, the header(s) will be displayed in the default size. See [AL\\_SetDefaultStyle](#).

**styleNum** — The **styleNum** is a font style code. By adding the codes together, you can combine styles. The numeric codes for **styleNum** are shown below:

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

**AL\_SetHdrStyle** can be used in the On load phase or in another phase (form event).

[AL\\_SetDefaultStyle](#) can be used to set the default values for the list data, the headers and the footers of all AreaList Pro areas.

Examples:

**AL\_SetHdrStyle**(elist;1;"Geneva";12;1) `Geneva 12 point bold, column 1

**AL\_SetHdrStyle**(elist;0;"Palatino";10;3) `Palatino 10 point bold italic, all columns

### AL\_GetHdrStyle

(areaRef:L; columnNumber:L; fontName:S; size:I; styleNum:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Desired list column
← fontName	string	Name of the font used
← size	integer	Size of the font
← styleNum	integer	Style of the font

**AL\_GetHdrStyle** returns the formatting options set using the [AL\\_SetHdrStyle](#) routine. For complete information on the values which are returned, please refer to the **AL\_SetHdrStyle** routine for parameter descriptions.

The following example will retrieve the information set using the **AL\_SetHdrStyle** routine:

```
C_LONGINT($fontSize;$fontStyle)
C_TEXT($fontName)
AL_GetHdrStyle(eList;1;$fontName;$fontSize;$fontStyle)
```

### AL\_SetFtrStyle

(areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column for which to set the footer style
→ fontName	string	Name of the font to use
→ size	integer	Size of the font
→ styleNum	integer	Style of the font

**AL\_SetFtrStyle** is used to control the appearance of the AreaList Pro column footers. The columns can be controlled individually or as a group.

**columnNumber** — This parameter specifies what column footer to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

**fontName** — Use this parameter to specify the font for the specified **columnNumber**. If not called, or the specified **fontName** is not found, the footer(s) will be displayed with the default font. See [AL\\_SetDefaultStyle](#). If the font specified by **fontName** is not installed, then the default font will be used.

## Configuring AreaList Pro Using Commands

**size** — Use this parameter to specify the font size for the specified **columnNumber**. If not called, the footer(s) will be displayed in the default size. See [AL\\_SetDefaultStyle](#).

**styleNum** — The **styleNum** is a font style code. By adding the codes together, you can combine styles. The numeric codes for **styleNum** are shown below:

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

**AL\_SetFtrStyle** can be used in the On load phase or in another phase (form event).

[AL\\_SetDefaultStyle](#) can be used to set the default values for the list data, the headers and the footers of all AreaList Pro areas.

---

## AL\_GetFtrStyle

(areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Desired list column
← fontName	string	Name of the font used
← size	integer	Size of the font
← styleNum	integer	Style of the font

**AL\_GetFtrStyle** returns the formatting options set using the [AL\\_SetFtrStyle](#) routine. For complete information on the values which are returned, please refer to the **AL\_SetFtrStyle** routine for parameter descriptions.

The following example will retrieve the information set using the **AL\_SetFtrStyle** routine:

```
C_LONGINT($fontSize;$fontStyle)
C_TEXT($fontName)
AL_GetFtrStyle(eList;1;$fontName;$fontSize;$fontStyle)
```

# Configuring AreaList Pro Using Commands

## AL\_SetStyle

(areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column for which to set the style
→ fontName	string	Name of the font to use
→ size	integer	Size of the font
→ styleNum	integer	Style of the font

**AL\_SetStyle** is used to control the appearance of the AreaList Pro columns. The columns can be controlled individually or as a group.

**columnNumber** — This parameter specifies what column to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

**fontName** — Use this parameter to specify the font for the specified **columnNumber**. If not called, or the specified **fontName** is not found, the column(s) will be displayed with the default font. See [AL\\_SetDefaultStyle](#). If the font specified by **fontName** is not installed, then the default font will be used.

**size** — Use this parameter to specify the font size for the specified **columnNumber**. If not called, the column(s) will be displayed in the default size. See [AL\\_SetDefaultStyle](#).

**styleNum** — The **styleNum** is a font style code. By adding the codes together, you can combine styles. The numeric codes for **styleNum** are shown below:

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

**AL\_SetStyle** can be used in the On load phase or in another phase (form event).

[AL\\_SetDefaultStyle](#) can be used to set the default values for the list data, the headers and the footers of all AreaList Pro areas.

Examples:

**AL\_SetStyle**(eList;0;"Geneva";9;0) `Geneva 9 plain, all columns

**AL\_SetStyle**(eList;4;"Helvetica";12;32) `Helvetica 12 point condensed, 4th column

### AL\_SetDefaultStyle

(selector:L; fontName:S; size:L; styleNum:L)

Parameter	Type	Description
→ selector	longint	Selector (area part for which to modify the default)
→ fontName	string	Name of the font to use
→ size	longint	Size of the font
→ styleNum	longint	Style of the font

**AL\_SetDefaultStyle** is used to control the list data, the headers and/or the footers appearance of all AreaList Pro areas to be created. Existing areas are not affected.

The appearance can eventually be modified for any part of an area using [AL\\_SetHdrStyle](#), [AL\\_SetFtrStyle](#) and [AL\\_SetStyle](#).

**selector** — indicates the area part for which to set the default appearance:

Constant	Value	Area Part
AL Style Header	1	Header
AL Style List	2	List data
AL Style Footer	3	Footer

**fontName** — Use this parameter to specify the font for the area part specified by **selector**.

**size** — Use this parameter to specify the font size for the area part specified by **selector**.

**styleNum** — The **styleNum** is a font style code. By adding the codes together, you can combine styles. The numeric codes for **styleNum** are shown below:

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

## Configuring AreaList Pro Using Commands

The defaults for the different area parts depend on the platform used.

MacOS defaults are:

Selector	Font	Size	Style
AL Style Header	Lucida Grande	13	0
AL Style List	Lucida Grande	11	0
AL Style Footer	Lucida Grande	11	0

Windows defaults are:

Selector	Font	Size	Style
AL Style Header	Tahoma	13	0
AL Style List	Tahoma	11	0
AL Style Footer	Tahoma	11	0

These values are initialized at startup from the STR# 15024 resource.

AreaList Pro versions prior to 8.1 used the following defaults: Geneva, 12, 0; Geneva, 10, 0; Geneva, 10, 0.

Example:

```
`Modify the default style for headers on Windows (Tahoma, 13, 0)
to Arial 12 bold
AL_SetDefaultStyle(AL_Style_Header;"Arial";12;1)
```

---

## AL\_GetStyle

(areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Desired list column
← fontName	string	Name of the font used
← size	integer	Size of the font
← styleNum	integer	Style of the font

**AL\_GetStyle** returns the formatting options set using the [AL\\_SetStyle](#) routine. For complete information on the values which are returned, please refer to the **AL\_SetStyle** routine for parameter descriptions.

The following example will retrieve the information set using the **AL\_SetStyle** routine:

```
C_LONGINT($fontSize;$fontStyle)
C_TEXT($fontName)
AL_GetStyle(eList;1;$fontName;$fontSize;$fontStyle)
```



### AL\_SetRowOpts

(areaRef:L; multiRows:I; allowNoSelection:I; dragRow:I; acceptDrag:I; moveWithData:I; disableRowHighlight:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ multiRows	integer	Single or multiple rows selection
→ allowNoSelection	integer	Allow no rows to be selected in single-row mode
→ dragRow	integer	Drag a row to this or another object
→ acceptDrag	integer	Accept drag from another AreaList Pro object
→ moveWithData	integer	Move row style and color with row
→ disableRowHighlight	integer	Disable highlighting of selected rows

**AL\_SetRowOpts** is used to control several AreaList Pro options pertaining to rows.

**multiRows:**

- 0** — allow only one row to be selected (default)
- 1** — allow the user to ctrl/command-click, shift-click, or drag to select multiple rows

In multi-rows mode, no rows are initially selected unless [AL\\_SetSelect](#) is used.

In single-row mode, the first row is selected unless [AL\\_SetLine](#) is used.

**allowNoSelection:**

- 0** — the user can not deselect a row (default)
- 1** — the user can ctrl/command-click to deselect a row in single-row mode

Regardless of the value of **allowNoSelection**, **AL\_SetLine** can be used with the **rowNumber** parameter set to 0 to set the selection to no rows.

**dragRow** — This parameter controls dragging of the area rows. See [Drag and Drop](#) and [Dragging Commands](#).

- 0** — do not allow a row to be dragged (default)

Values 1, 2, and 3 enable a row to be dragged while the option/alt key is pressed:

- 1** — allow a row to be dragged within, but not out of the AreaList Pro object
- 2** — allow a row to be dragged out of, but not within the AreaList Pro object
- 3** — allow a row to be dragged both within and out of the AreaList Pro object

Values 4, 5, and 6 enable a row to be dragged without any modifier key:

- 4** — allow a row to be dragged within, but not out of the AreaList Pro object
- 5** — allow a row to be dragged out of, but not within the AreaList Pro object
- 6** — allow a row to be dragged both within and out of the AreaList Pro object

## Configuring AreaList Pro Using Commands

If a row is dragged without any modifier key, dragging to select multiple rows will not work.

If the row is dragged to another position within the list, AreaList Pro will automatically rearrange the whole list. If the row is dragged out of the list to another AreaList Pro object, it is up to you to remove and insert row(s) as necessary.

**acceptDrag:**

- 0** — this AreaList Pro object will not accept a row (default)
- 1** — this AreaList Pro object will accept a row dragged from another AreaList Pro object

**moveWithData** — This parameter is used with various formatting commands such as [AL\\_SetCellEnter](#), [AL\\_SetRowStyle](#), [AL\\_SetRowColor](#), [AL\\_SetRowRGBColor](#), [AL\\_SetCellStyle](#), [AL\\_SetCellColor](#) and [AL\\_SetCellRGBColor](#):

- 0** — the row style and color information will not move with the row
- 1** — the row style and color information will move with the row whenever the AreaList Pro object is sorted or a row is dragged within the list (default)

This parameter is ignored when displaying fields. See [Drag and Drop](#) and [Dragging Commands](#) for more information.

**disableRowHighlight:**

- 0** — all selected rows will be highlighted when selected (default)
- 1** — no rows will be highlighted when selected

When **disableRowHighlight** is set to 1, no rows will be highlighted if the user selects them or if they are selected by calling the commands **AL\_SetLine** or **AL\_SetSelect**. AreaList Pro will still maintain a list of the selected rows, even though they will not be highlighted. Thus the commands **AL\_GetClickedRow** or **AL\_GetSelect** will still return the correct selected row(s). This parameter is especially useful if you want to have a different way of showing selected rows such as by having a column of check marks or bullets.

**AL\_SetRowOpts** can be used in the [On load](#) phase or in another phase (form event).

Examples:

`Setup the list for single-row selection, allow the user to select no rows, don't allow the user to drag rows, don't accept a drag from another AreaList Pro object, don't move the row style and color info with the row, don't disable row highlighting

**AL\_SetRowOpts**(eList;0;1;0;0;0;0)

`Setup the list for multi-rows selection, require one row selection, allow the user to option / alt-drag rows only within the list, accept a drag from another AreaList Pro object, move the row style and color info with the row, disable row highlighting

**AL\_SetRowOpts**(eList;1;0;1;1;1;1)

`Setup the list for single-row selection, require one row selection, allow the user to drag rows within the list and out of the list without the option / alt key, accept a drag from another AreaList Pro object, move the row style and color info with the row, disable row highlighting

**AL\_SetRowOpts**(eList;0;0;6;1;1;1)

### AL\_GetRowOpts

(areaRef:L; multiRows:I; allowNoSelection:I; dragRow:I; acceptDrag:I; moveWithData:I; disableRowHighlight:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← multiRows	integer	Single or multiple rows selection
← allowNoSelection	string	Allow no rows to be selected in single-row mode
← dragRow	integer	Drag a row to this or another object
← acceptDrag	integer	Accept drag from another AreaList Pro object
← moveWithData	integer	Move row style and color with row
← disableRowHighlight	integer	Disable highlighting of selected rows

**AL\_GetRowOpts** will return the current settings configured using [AL\\_SetRowOpts](#). For complete details about return values, please see the **AL\_SetRowOpts** routine for possible configuration settings.

### AL\_SetColOpts

(areaRef:L; allowColumnResize:I; automaticResize:I; allowColumnLock:I; hideLastColumns:I; displayPixelWidth:I; dragColumn:I; acceptDrag:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ allowColumnResize	integer	User resizable columns
→ automaticResize	integer	Automatically resize columns when events occur in the area
→ allowColumnLock	integer	Allow user to lock columns
→ hideLastColumns	integer	Number of columns from the right to hide
→ displayPixelWidth	integer	Display column widths
→ dragColumn	integer	Drag a column to this or another object
→ acceptDrag	integer	Accept drag from another AreaList Pro object

**AL\_SetColOpts** is used to control several AreaList Pro options pertaining to columns.

**allowColumnResize** — This parameter controls whether the user can resize column by clicking on the dividing line between column headers:

- 0 — do not allow the user to resize columns
- 1 — allow the user to resize columns (default)

When the **hideHeaders** parameter of **AL\_SetMiscOpts** is set to 1 (headers are hidden), **allowColumnResize** is set to 0 internally by AreaList Pro.

## Configuring AreaList Pro Using Commands

The \$2 event code returned to the callback method (or [AL\\_GetLastEvent](#) command, formerly [ALProEvt variable](#)) will be set to -3 if the user resizes a column (see [Determining the User's Action on an AreaList Pro Object](#)). You can get the column widths using [AL\\_GetWidths](#).

**automaticResize:**

- 0** — No columns will be resized (default).
- 1** — Whenever an array or field command is called while the area is displayed, the columns will be resized to the last widths passed using [AL\\_SetWidths](#). If any column widths are 0, then AreaList Pro will automatically calculate the width based upon the contents of the column.

**allowColumnLock:**

- 0** — disables the column lock area, which prevents the user from modifying the number of locked columns
- 1** — enables the column lock area of the AreaList Pro object, allowing the user to modify the number of locked columns (default)

The \$2 event code returned to the callback method (or [AL\\_GetLastEvent](#) command, formerly [ALProEvt variable](#)) will be set to -4 if the user changes the column lock position (see [Determining the User's Action on an AreaList Pro Object](#)).

You can determine the current column lock position using [AL\\_GetColLock](#).

**hideLastColumns** — This parameter specifies the number of columns from the right to not display:

- 0** — forces the display of all columns (default)
- 1 to (number of columns -1)** — number of columns to hide

This parameter is used when an ID column is needed for **SEARCH** purposes after the list is displayed, but you don't want to clutter the display with the ID values. You would pass the ID array as the last array to [AL\\_SetArraysNam](#), and hide the last column using this parameter with a value of one. Any pre-sort or user-sort will include the hidden column(s), to keep the values in all the columns "lined-up." If the number of columns passed to AreaList Pro is less than or equal to the value specified by **hideLastColumns**, then only the first column will be displayed.

**displayPixelWidth** — Used during development to allow you to easily determine what pixel width looks best for each column. When this option is enabled, a button in the lower right area of the AreaList Pro object is enabled to toggle the headers between displaying pixel widths and the actual header values. When AreaList Pro is initially displayed, the column headers are shown. Click on the button to toggle the headers to display the pixel width.

- 0** — turns the pixel width display off and disables the button (default)
- 1** — column headers display the width in pixels of each column, and are updated after the user resizes the column

When pixel widths are displayed in the headers of the AreaList Pro area, the cursor will change to display a pixel count when it is over the AreaList Pro area. If the cursor is moved over one of the rows in the area and clicked, the count shown in the pointer will be updated.

This value is the necessary height of the AreaList Pro object to allow the row clicked on to be the bottom one displayed. This feature is disabled whenever the column widths are not displayed. Please read the section [Column Widths](#) for more information.

## Configuring AreaList Pro Using Commands

**dragColumn** — This parameter controls if, and how, columns may be dragged. See [Drag and Drop and Dragging Commands](#).

- 0** — do not allow a column to be dragged (default)
- 1** — allow a column to be dragged within, but not out of the AreaList Pro object
- 2** — allow a column to be dragged out of, but not within the AreaList Pro object
- 3** — allow a column to be dragged both within and out of the AreaList Pro object

If the **userSort** option of **AL\_SetSortOpts** is disabled, column dragging will begin immediately after the user clicks in the column header, and an outline of the column will appear.

If user sorting is enabled, the drag begins when the pointer is moved 20 pixels outside of the column to the left or right, or 30 pixels above or below the header area.

It is up to you to keep track of the new position of the columns: dragging the first column to the right will cause the second column to become the first. Future calls to AreaList Pro code should take these changes into account.

**acceptDrag** — This parameter controls whether columns may be dragged into the AreaList Pro object **areaRef**:

- 0** — this AreaList Pro object will not accept a column (default)
- 1** — this AreaList Pro object will accept a column dragged from another AreaList Pro object

**AL\_SetColOpts** can be used in the On load phase or in another phase (form event).

Examples:

`Allow user to resize columns, don't resize columns while the area is displayed, allow column lock, hide the last two columns, disable the pixel width display, don't allow or accept column dragging

**AL\_SetColOpts**(eList;1;0;1;2;0;0;0)

`Don't allow user to resize columns, resize columns while the area is displayed, allow column lock, don't hide any columns, enable the pixel width display, don't allow column dragging, but accept dragged columns

**AL\_SetColOpts**(eList;0;1;1;0;1;0;1)

## Configuring AreaList Pro Using Commands

### AL\_GetColOpts

(areaRef:L; allowColumnResize:I; automaticResize:I; allowColumnLock:I; hideLastColumns:I; displayPixelWidth:I; dragColumn:I; acceptDrag:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← allowColumnResize	integer	User resizable columns
← automaticResize	integer	Automatically resize columns when events occur in the area
← allowColumnLock	integer	Allow user to lock columns
← hideLastColumns	integer	Number of columns from the right to hide
← displayPixelWidth	integer	Display column widths
← dragColumn	integer	Drag a column to this or another object
← acceptDrag	integer	Accept drag from another AreaList Pro object

**AL\_GetColOpts** will return the current settings configured using [AL\\_SetColOpts](#). For complete details about return values, please see the **AL\_SetColOpts** routine for possible configuration settings.

### AL\_SetCellOpts

(areaRef:L; cellSelection:I; moveWithData:I; optimization:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellSelection	integer	Cell selection mode
→ moveWithData	integer	Move cell attributes with data
→ optimization	integer	Optimize cell attribute allocation

**AL\_SetCellOpts** is used to set options specific to cells.

**cellSelection:**

- 0** — row selection is enabled according to the **multiRows** option of [AL\\_SetRowOpts](#) (default)
- 1** — only one cell at a time may be selected (single cell selection)
- 2** — several cells may be selected, contiguous or discontinuous (multiple cells selection)
- 3** — row selection is enabled according to the **multiRows** option of **AL\_SetRowOpts** and the multiple rows keyboard scrolling is active as described below

## Configuring AreaList Pro Using Commands

### Up/Down Arrow Keys

When AreaList Pro has been configured to allow multiple rows selection and the user presses the up or down Arrow keys, the following conditions apply:

- when the up Arrow key is pressed, the row prior to the first highlighted row will be selected and will be the new active row
- when the down Arrow key is pressed, the row after the last highlighted row will be selected and will be the new active row

This interface is off by default (for backwards compatibility with previous versions) and may be activated using the `cellSelection` parameter of **AL\_SetCellOpts**.

The following parameter will activate the keyboard scrolling options when using the multi-rows selection option:

**AL\_SetCellOpts**(eList;3;...) `turn on enhanced Arrow key support

---

*When cellSelection is set to a value other than 0, row dragging is disabled.*

---

**moveWithData** — This parameter is used with various formatting commands such as [AL\\_SetCellEnter](#), [AL\\_SetRowStyle](#), [AL\\_SetRowColor](#), [AL\\_SetRowRGBColor](#), [AL\\_SetCellStyle](#), [AL\\_SetCellColor](#) and [AL\\_SetCellRGBColor](#):

- 0** — cell attributes will not move
- 1** — cell attributes (not including cell selection) will move with the cell after sorting, row dragging, or column dragging (default)

This parameter is ignored when displaying fields. See [Drag and Drop](#) and [Dragging Commands](#) for more information.

**optimization** — The default is 1.

A value of 1 means that the block used to store the cell attributes (per row) is grown a small chunk at a time. A value of 5 means that the block used to store the cell attributes is grown a large chunk at a time. Thus a lower number means that setting cell attributes may be slower but will (potentially) require less memory. Conversely, a higher number means that setting cell attributes may be faster but requires more memory.

**optimization** should not be set above 1 unless the number of columns in the AreaList Pro object is greater than 10 and a large percentage of the cells will have their cell attributes set.

Example:

**AL\_SetCellOpts**(eList;1;1;1) `single cell selection only, move data with cells, normal optimization



## Configuring AreaList Pro Using Commands

### AL\_GetCellOpts

(areaRef:L; cellSelection:I; moveWithData:I; optimization:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← cellSelection	integer	Cell selection mode
← moveWithData	integer	Move cell attributes with data
← optimization	integer	Optimize cell attribute allocation

**AL\_GetCellOpts** will return the current values set by [AL\\_SetCellOpts](#) (or default values if this routine has not been called). For complete details about return values, please see the **AL\_SetCellOpts** routine for possible configuration settings.

### AL\_SetInterface

(areaRef:L; appearance:L; sortIndicator:L; useEllipsis:L; ignoreMenuMeta:L; clickDelay:L; allowPartialRow:L; useOldPopup:L; entryControls:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← → appearance	longint	Column style
← → sortIndicator	longint	Sort Icon
← → useEllipsis	longint	Use ellipsis
← → ignoreMenuMeta	longint	Enable/disable meta characters in AreaList Pro enterable popup controls
← → clickDelay	longint	Click delay to initiate data entry
← → allowPartialRow	longint	Allow partial rows to be displayed
← → useOldPopup	longint	Use old date and time popups
← → entryControls	longint	Use inline controls for date and time data entry

**AL\_SetInterface** provides the ability to customize the default appearance settings of an AreaList Pro area.

*In all parameters except areaRef, passing a value of -1 through a variable will return the current setting (←) in this variable and do nothing to the appearance.*

## Configuring AreaList Pro Using Commands

If this routine is not called, AreaList Pro the following defaults will be used:

Parameter	Value	Description
appearance	0	Appearance will use the default platform interface based on the current client (machine) using AreaList Pro
sortIndicator	0	Sort Icon will appear in column header
useEllipsis	0	Ellipsis characters will not be used by default
ignoreMenuMeta	0	Meta characters enabled
clickDelay	60	AreaList Pro will wait one second before automatically initiating data entry
allowPartialRow	0	Don't display partial row, area will be resized vertically to only show full rows
useOldPopup	0	Use new date and time popups
entryControls	0	Don't use inline controls for date and time data entry, use plain text instead

**areaRef** — AreaList Pro area reference (or -1 for global setting).

---

*You can use this routine to provide global configuration for all AreaList Pro areas in your database by passing an area reference of (-1) as the first parameter. This does not affect already created areas.*

---

**appearance** — Instructs AreaList Pro to use the defined appearance setting, regardless of the current OS platform.

Constant	Value	Description
AL Default Interface	0	Default platform appearance
AL Platinum Interface	1	Platinum (Mac OS9) appearance
AL Force OSX Interface	2	Force Mac OSX appearance
AL Force XP Interface	3	Force Windows XP appearance
AL Force Vista Interface	4	Force Vista appearance

---

*This parameter will be ignored if the useModernLook parameter of [AL\\_SetMiscOpts](#) is set to a value of 0.*

---

**sortIndicator** — Determines the location of the sort icon:

- 0** — sort icon in header (OSX/XP only)
- 1** — sort icon above scrollbar

**useEllipsis** — Determines if auto-ellipsis is used:

- 0** — don't use ellipsis in header and column data
- 1** — use ellipsis in header and column data (in the center for right aligned text)
- 2** — use ellipsis in header and column data (on the left side for right aligned text)

## Configuring AreaList Pro Using Commands

The following example will force Windows XP, place the sort icon above the scrollbar and disable the ellipsis:

**AL\_SetInterface**(eList;3;1;0)

If this routine were called on Mac OSX, the headers would have a WinXP interface. Conversely, you could do the same thing (for Mac OSX appearance on Windows).

**ignoreMenuMeta** — Provides a global interface for disabling meta characters in AreaList Pro enterable popup controls:

**0** — meta characters enabled

**1** — meta characters disabled

**clickDelay** — Provides the number of ticks (60 ticks = 1 second) which AreaList Pro will wait before automatically initiating data entry (see [AL\\_SetEntryOpts](#) and [AL\\_SetEnterable](#) for additional options):

**0** — inactive

**-2** — use the system's double click time

**1 to 300** — number of ticks

**allowPartialRow** — Instructs AreaList Pro to display partial rows, thus causing AreaList Pro to draw in the exact area rectangle defined in form:

**0** — don't display partial row, area will be resized vertically to only show full rows (default)

**1** — allow display of partial rows (area will not be resized)

**useOldPopup** — Instructs AreaList Pro to display the old style time and date popups or the new one (see [Data Entry Using Popups](#) for more details):

**0** — use new date and time popups

**1** — use old date and time popups

**entryControls** — Determines if time and/or date data is entered as plain text or through inline controls (see [Data Entry Using Inline Controls](#) for more details):

**0** — use plain text for both times and dates

**1** — use inline controls for times and plain text for dates

**2** — use plain text for times and inline controls for dates

**3** — use inline controls for both times and dates

The example below illustrates the combination of **AL\_SetInterface**, [AL\\_SetMiscOpts](#) and [AL\\_GetMiscOpts](#).

## Configuring AreaList Pro Using Commands

Here is the On load phase of the **eList** AreaList Pro area's object method:

```
$result:=AL_SetArraysNam(eList;1;3;"rConstant";"rValue";"rDescription")
```

```
AL_SetHeaders(eList;1;3;"Constant";"Value";"Description")
```

```
AL_SetFormat(eList;2;"0";2;0;0;0)
```

```
AL_SetScroll(eList;-3;-3) `no scrollbars
```

```
AL_GetMiscOpts(eList;0;vAreaSelected;"";0;cModern)
```

```
`get default values (vAreaSelected=0 / cModern=1)
```

vAreaSelected:=2 `a 2-pixel wide border will be drawn around the plug-in area when it is selected

```
AL_SetMiscOpts(eList;0;vAreaSelected;"";0;cModern)
```

vAppearance:=1 `get the default value

```
AL_SetInterface(eList;vAppearance;-1;-1;-1;-1;-1) `now vAppearance is set to default (0)
```

```
$result:=AL_SetEventCallback(eList;"mCallBack";2)
```

Our list displays the **AL\_SetInterface** constants, row 1 being Default Interface (value 0). Thus the constant value is the row number minus 1.

The *mCallBack* project method is as follows:

```
`Event Callback
```

vAppearance:=**AL\_GetLine**(\$1)-1 `get the row that was selected minus 1 = constant value

```
AL_SetInterface($1;vAppearance;-1;-1;-1;-1;-1) `set the interface according to the selected  
appearance, do nothing to the other parameters
```

```
$0:=0 `return zero = do not execute object method and form method
```

Here is the result on Mac OSX with an interface set to XP, the **areaSelected** parameter of [AL\\_SetMiscOpts](#) set to 2 and **useModernLook** to 1:

Constant	Value	Description
AL Default Interface	0	Default Platform Appearance
AL Platinum Interface	1	Platinum (Mac OS9) Appearance
AL Force OSX Interface	2	Force Mac OSX Appearance
AL Force XP Interface	3	Force Windows XP Appearance
AL Force Vista Interface	4	Force Vista Appearance

### AL\_SetMiscOpts

(areaRef:L; hideHeaders:I; areaSelected:I; postKey:S; showFooters:I; useModernLook:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ hideHeaders	integer	Hide the column headers
→ areaSelected	integer	Visual cue that the plug-in area is selected
→ postKey	string	String character to post to execute method
→ showFooters	integer	Show the column footers
→ useModernLook	integer	Modern or traditional appearance

**AL\_SetMiscOpts** is used to control several AreaList Pro options.

**hideHeaders:**

- 0 — the column headers will be displayed (default)
- 1 — the column headers will not be displayed

When **hideHeaders** is 1, the **allowColumnResize** parameter of **AL\_SetColOpts** is set to 0 internally by AreaList Pro.

**areaSelected** — This parameter controls how the AreaList Pro object is displayed when it is “selected” (i.e. the active layout object):

- 0 — no indication will be given to the user that the plug-in area is selected (default)
- 1 — a 2-pixel wide border will be drawn around the plug-in area when it is selected
- 2 — a selection rectangle will be drawn around the plug-in area when it is selected
- 3 — a 3D frame will be drawn around the object when it is selected

**postKey** — One character string. AreaList Pro causes the method of an AreaList Pro plug-in area and the form method to run by posting a keyboard event to 4D’s event queue. This parameter is used to specify what character to post. The default is the backslash character (“\”). Please read the section [AreaList Pro’s PostKey](#) for more information. This can be ignored since 4D 2004.

**showFooters** — This parameter controls whether footers are displayed for the AreaList Pro object **areaRef**. Footers are displayed using **AL\_SetFooters**.

- 0 — footers will not be displayed (default)
- 1 — footers will be displayed below each column

See [Footers](#), [AL\\_SetFooters](#) and [AL\\_SetFtrStyle](#) for more information about footers.

## Configuring AreaList Pro Using Commands

`useModernLook`:

- 0** — the “traditional” look will be used
- 1** — a modern look will be used (default)

When `useModernLook` is set to a value of 1, the appearance of the AreaList Pro object will be set according to the current OS, or to the `appearance` parameter of [AL\\_SetInterface](#).

**`AL_SetMiscOpts`** can be used in the On load phase or in another phase (form event).

Examples:

‘Don’t hide the headers, show the area selected in 3D frame, use the default postKey, don’t show footers, use modern appearance

**`AL_SetMiscOpts`**(eList;0;1;"";0;1)

‘Hide the headers, don’t show the area selected cue, use open bracket for the postKey, show footers, use traditional appearance

**`AL_SetMiscOpts`**(eList;1;0;"[";1;0)

See also the example provided for [AL\\_SetInterface](#) to illustrate the combination of this command with **`AL_SetMiscOpts`** and [AL\\_GetMiscOpts](#).

---

## AL\_GetMiscOpts

(areaRef:L; hideHeaders:I; areaSelected:I; postKey:S; showFooters:I; useModernLook:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← hideHeaders	integer	Hide the column headers
← areaSelected	integer	Visual cue that the plug-in area is selected
← postKey	string	String character to post to execute method
← showFooters	integer	Show the column footers
← useModernLook	integer	Modern or traditional appearance

**`AL_GetMiscOpts`** will return the current settings configured using [AL\\_SetMiscOpts](#). For complete details about return values, see the **`AL_SetMiscOpts`** routine for possible configuration settings.

### AL\_SetMiscColor

(areaRef:L; selector:I; alpColor:S; 4dColor:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ selector	integer	Select which part to apply the color to
→ alpColor	string	Color from AreaList Pro's palette
→ 4dColor	integer	Color from 4D's palette

**AL\_SetMiscColor** is used to set the color of miscellaneous parts of an AreaList Pro object.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

**selector:**

- 0** — The background color of the area in the header above the vertical scrollbar.  
This area only exists if the header and the vertical scrollbar are shown.
- 1** — The background color of the area in the footer below the vertical scrollbar.  
This area only exists if the footer and the vertical scrollbar are shown.
- 2** — The background color of the area to the left of the horizontal scrollbar.  
This area only exists if the horizontal scrollbar is shown and at least one column is locked.
- 3** — The background color of the area to the right of the horizontal scrollbar.  
This area only exists if the horizontal scrollbar and the vertical scrollbar are shown.

**alpColor** — Name of the color in AreaList Pro's palette. This will be the color for the part specified by **selector**. If the name is not in AreaList Pro's palette or it is a null (empty) string, then **4dColor** will be used.

**4dColor** — 1 to 256. The color at this position in 4D's palette will be used for the color for the part specified by **selector**.

Examples:

`Light gray for the area in the header above the vertical scrollbar

**AL\_SetMiscColor**(eList;0;"Light Gray";0)

`13th color from 4D's palette for the area to the left of the horizontal scrollbar

**AL\_SetMiscColor**(eList;2;"";13)



### AL\_SetMiscRGBColor

(areaRef:L; selector:L; red:L; green:L; blue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ selector	longint	Select which part to apply the color to
→ red	longint	Red
→ green	longint	Green
→ blue	longint	Blue

**AL\_SetMiscRGBColor** provides the ability to define miscellaneous color attributes using the associated RGB values. This routine is similar to [AL\\_SetMiscColor](#).

**red** — Desired red component in RGB color pattern.

**green** — Desired green component in RGB color pattern.

**blue** — Desired blue component in RGB color pattern.

### AL\_SetCopyOpts

(areaRef:L; includeHiddenCols:I; fieldDelimiter:S; recordDelimiter:S; fieldWrapper:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ includeHiddenCols	integer	Include hidden columns
→ fieldDelimiter	string	Field separator for Edit menu copy
→ recordDelimiter	string	Record separator for Edit menu copy
→ fieldWrapper	string	Field wrapper for Edit menu copy

**AL\_SetCopyOpts** is used to control several AreaList Pro options pertaining to copying the selected row(s) when “Copy” is selected from the Edit menu. Because of limitations of the clipboard, picture columns cannot be copied to the clipboard; a blank field will be copied instead.

For greater control over the Edit menu, you can use the [AL\\_SetEditMenuCallback](#) interface.

## Configuring AreaList Pro Using Commands

**includeHiddenCols:**

- 1** — any values in hidden columns will be included when the user uses the Edit menu Copy command
- 0** — any values in hidden columns will not be included when the user uses the Edit menu Copy command (default)

**fieldDelimiter** — One character string. The delimiter used to separate fields when the user copies selected rows to the clipboard. Default is the tabulation character (ASCII 9).

**recordDelimiter** — One character string. The delimiter used to separate rows when the user copies selected rows to the clipboard. Default is the carriage return character (ASCII 13).

**fieldWrapper** — One character string. The character used to “wrap” fields when the user copies selected rows to the clipboard. This character will be placed both before and after each field. If **fieldWrapper** is the null (empty) string, then no character will wrap the fields. The default is that no character will wrap the fields.

**fieldWrapper** will be especially useful on Windows because programs such as Excel or Works expect text to be pasted in with commas separating, and quotes wrapping the fields.

**AL\_SetCopyOpts** can be used in the On load phase or in another phase (form event).

Examples:

‘Include hidden columns in Edit menu Copy, use the default Field and Record delimiters for Edit menu Copy

**AL\_SetCopyOpts** (eList;1;"","")

‘Don’t include hidden columns in Edit menu Copy, use different Field and Record delimiters for Edit menu Copy

**AL\_SetCopyOpts** (eList;0;**Char** (241);**Char** (242))

---

## AL\_GetCopyOpts

(areaRef:L; includeHiddenCols:I; fieldDelimiter:S; recordDelimiter:S; fieldWrapper:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← includeHiddenCols	integer	Include hidden columns
← fieldDelimiter	string	Field separator for Edit menu copy
← recordDelimiter	string	Record separator for Edit menu copy
← fieldWrapper	string	Field wrapper for Edit menu copy

**AL\_GetCopyOpts** will return the current values as defined by [AL\\_SetCopyOpts](#).

### AL\_SetSortOpts

(areaRef:L; automaticSort:I; userSort:I; allowSortEditor:I; sortEditorPrompt:S;  
showSortOrder:I; showSortDirIndicator:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ automaticSort	integer	Automatically sort
→ userSort	integer	Allow user to sort
→ allowSortEditor	integer	Allow user to sort with editor
→ sortEditorPrompt	string	Set the prompt of the Sort Editor
→ showSortOrder	integer	Show the current sort order in the Sort Editor
→ showSortDirIndicator	integer	Show the current sort direction in upper right corner

**AL\_SetSortOpts** is used to control several AreaList Pro options pertaining to sorting.

**automaticSort:**

- 1** — whenever an array or field command is called while the area is displayed, the columns will be automatically sorted based upon the current sort order
- 0** — no sorting will be done automatically (default)

**userSort:**

- 0** — Disable the user sort buttons in the column headers.
- 1** — Enable the user sort buttons in the column headers (default). The sort buttons will highlight when clicked, and the columns will be sorted based on the values in the column which was clicked. The AreaList Pro event callback (or area/form method) will run, with a \$2 event code of -1 returned to the callback method (or [AL\\_GetLastEvent](#) command, formerly **ALProEvt** variable).
- 2** — Bypass the user sort buttons in the column headers. The sort buttons will highlight when clicked, but no sort will be performed. The AreaList Pro event callback (or area/form method) will run, with a \$2 event code of -1 returned to the callback method (or **AL\_GetLastEvent** command, formerly **ALProEvt** variable). This allows you to procedurally check for a click of a sort button by the user and perform your own sort action.
- 3** — Enable the user sort buttons for indexed fields only. If the field in the column is not indexed, the sort button will highlight when clicked, but no sort will be performed. If the field in the column is indexed, the fields will be sorted based on the values in the column which was clicked. The AreaList Pro event callback (or area/form method) will run, with a \$2 event code of -1 returned to the callback method (or **AL\_GetLastEvent** command, formerly **ALProEvt** variable). If arrays, not fields, are displayed in the object then all of the sort buttons will be enabled.

## Configuring AreaList Pro Using Commands

In the following situations, the column header will highlight, but no sort will occur, no event callback method will be called and the method for the AreaList Pro area/form will not run:

- if the value of **userSort** is 1, 2 or 3, and the column contains a picture column
- if the value of **userSort** is 1 or 3, and the column contains a field from a related one table

If the value of **userSort** is 2, and the column contains a field from a related one table, the column header will highlight, but no sort will occur, the event callback method will be called if defined, and/or the method for the AreaList Pro area/form will run.

When the user sort is bypassed by setting **userSort** to 2, [AL\\_GetSort](#) is still used to get the column header that was clicked on.

### **allowSortEditor:**

- 1** — the user can ctrl/command-click in the header to display the AreaList Pro Sort Editor
- 0** — the user is not able to display the Sort Editor (default)

If the value of **allowSortEditor** is 1, the following actions will trigger the display of the AreaList Pro Sort Editor:

- Windows ctrl-click
- MacOS command-click

In addition, similar actions such as right-click or MacOS ctrl-click can trigger an event report without displaying the AreaList Pro Sort Editor. See [Ctrl/command-click in the Column Header Event](#).

The AreaList Pro Sort Editor can also be displayed with [AL\\_ShowSortEd](#).

[AL\\_SetSortedCols](#) provides the ability to customize the default list of sorted columns.

[AL\\_GetSortedCols](#) returns the current sort columns as displayed in the Sort Editor.

**sortEditorPrompt** (optional) — This is the prompt that will be displayed at the top of the AreaList Pro Sort Editor. The default is "Select columns to sort".

The Sort Editor prompt can also be modified using [AL\\_SetSortEditorParams](#).

### **showSortOrder:**

- 1** — the current sort order will be displayed in the sort order list whenever the AreaList Pro Sort Editor is displayed
- 0** — the sort order list will be empty whenever the AreaList Pro Sort Editor is displayed (default)

### **showSortDirIndicator:**

- 1** — a sort direction indicator will be displayed in the upper right corner above the vertical scrollbar
- 0** — no sort direction indicator will be displayed (default)

## Configuring AreaList Pro Using Commands

Displaying the sort indicator requires the header and the vertical scrollbar to be displayed. Please read the section [AL\\_SetScroll](#) for more information.

[AL\\_SetHeaderOptions](#) provides the ability to customize the interface over the scrollbars (sort area). You can customize the icon which is displayed using a “cicn” or “PICT” resource, or an item from the 4D Picture Library.

When the user clicks the sort direction indicator, the sort direction of the primary sort level will be reversed and the list will be re-sorted. The AreaList Pro event callback (or area/form method) will run, with a \$2 event code of -1 returned to the callback method (or [AL\\_GetLastEvent](#) command, formerly **ALProEvt variable**), the same as if a sort button in the header was clicked.

**AL\_SetSortOpts** can be used in the On load phase or in another phase (form event).

Examples:

```
`Don't automatically sort, allow user to sort with buttons, allow user to invoke Sort Editor,  
display the default Sort Editor prompt, don't show the current sort order in the Sort Editor,  
show the sort direction indicator
```

```
AL_SetSortOpts(eList;0;1;1;"";0;1)
```

```
`Automatically sort, don't allow user to sort with buttons, allow user to invoke Sort Editor, change  
the Sort Editor prompt, show the current sort order in the Sort Editor, don't show the sort direction indicator
```

```
AL_SetSortOpts(eList;1;0;1;"People Sort Order";1;0)
```

---

## AL\_SetSortEditorParams

(areaRef:L; windowTitle:S; prompt:S; labelList:X; columnNumberList:X) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ windowTitle	string	Window title
→ prompt	string	Editor prompt
→ labelList	text array	Sort column labels
→ columnNumberList	longint array	Sort column numbers
← resultCode	longint	Result code

**AL\_SetSortEditorParams** provides the ability to customize the appearance of available sort items when displaying the AreaList Pro Sort Editor.

**windowTitle** — Sets the Sort Editor window title (default “Sort Options”). Passing a null string (no value) will tell AreaList Pro to use the default (or current) window title.

## Configuring AreaList Pro Using Commands

**prompt** — Sets the Sort Editor dialog prompt (default “Select columns to sort”). Passing a null string (no value) will tell AreaList Pro to use the default (or current) dialog prompt.

**labelList** — Sets the displayed names of the columns that will be available for sorting, which can be the header names or customized labels.

**columnNumberList** — Set the column numbers that will be available for sorting. If not supplied, all displayed columns will be used.

**resultCode** — Returns a valid AreaList Pro result code.

The following example will define the Sort Editor to display two columns (column 1 and 3) in the Sort Editor, overriding the default settings. In addition, we’ll override the default Sort Editor prompt, and use the default window title:

```
ARRAY TEXT (atColList;2)
ARRAY LONGINT (aiColList;2)
atColList{1}:="First"
atColList{2}:="Third"
aiColList{1}:=1 `first column
aiColList{2}:=3 `third column
$ret:=AL_SetSortEditorParams (eList;"","Select Column";atColList;aiColList)
```

---

### AL\_GetSortEditorParams

(areaRef:L; windowTitle:S; prompt:S; headerList:X; sortList:X) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← windowTitle	string	Window title
← prompt	string	Editor prompt
← labelList	text array	Sort column labels
← columnNumberList	longint array	Sort column numbers
← resultCode	longint	Result code

**AL\_GetSortEditorParams** provides the ability to retrieve the current properties of the AreaList Pro Sort Editor. If you have not previously customized the display properties, the default settings will be returned. See [AL\\_SetSortEditorParams](#) for information on setting the Sort Editor attributes.

## Configuring AreaList Pro Using Commands

The following example will assume you have not previously called the **AL\_SetSortEditorParams** routine and will return all the default settings:

```
C_TEXT (sAL_WindowTitle)
C_TEXT (sAL_SortEditorPrompt)
ARRAY TEXT (atAL_SortNames;0)
ARRAY LONGINT (aiAL_SortColumnOrder;0)
$ret:=AL_GetSortEditorParams (eList;sAL_WindowTitle;sAL_SortEditorPrompt;atAL_SortNames;
    aiAL_SortColumnOrder;0)
```

---

### AL\_SetSortedCols

(areaRef:L; sortList:X) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ sortList	longint array	Sort column numbers
← resultCode	longint	Result code

**AL\_SetSortedCols** provides the ability to customize the default list of sorted columns.

If you don't customize the sort list, the Sort Editor will use the current sort order by default and if you have previously displayed the Sort Editor and defined more than one column, they will be displayed when the Sort Editor is displayed again.

By default, the Sort Editor will always use what the user has selected in the sort column and previous Sort Editor actions, unless you override the column list procedurally using **AL\_SetSortedCols**.

**sortList** — A valid 4<sup>th</sup> Dimension longint array which will contain the column(s) you wish to display in the sorted column list. If you define a column that is outside the displayed columns, nothing will be displayed.

- if you wish to have the column sorted in ascending order, pass a positive value
- if you wish to have the column sorted in descending order, pass a negative value

**resultCode** — Returns a valid AreaList Pro result code.

The following example will configure the Sort Editor to display the third column (ascending) and fifth column (descending) in the sorted list:

```
ARRAY LONGINT (aiAL_SortCols;2)
aiAL_SortCols{1}:=3 `include the 3rd column in sort list
aiAL_SortCols{2}:=5 `include the 5th column in descending order
$ret:=AL_SetSortedCols (eList;aiAL_SortCols)
```

[AL\\_GetSortedCols](#) returns the current sort columns as displayed in the Sort Editor.



### AL\_SetForeColor

(areaRef:L; columnNumber:I; alpHdrForeColor:S; 4dHdrForeColor:I; alpListForeColor:S; 4dListForeColor:I; alpFtrForeColor:S; 4dFtrForeColor:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column number
→ alpHdrForeColor	string	Header foreground color from Arealist Pro's palette
→ 4dHdrForeColor	integer	Header foreground color from 4D's palette
→ alpListForeColor	string	List foreground color from Arealist Pro's palette
→ 4dListForeColor	integer	List foreground color from 4D's palette
→ alpFtrForeColor	string	Footer foreground color from Arealist Pro's palette
→ 4dFtrForeColor	integer	Footer foreground color from 4D's palette

**AL\_SetForeColor** is used to specify the foreground colors for a column header, a list area column, and a column footer.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

**columnNumber** — The column for which to set the foreground color. Use a value of zero (0) for **columnNumber** to apply the parameters to all columns.

**alpHdrForeColor** — Name of the color in AreaList Pro's palette. This will be the foreground color for the column header. If the name is not in AreaList Pro's palette or it is a null string, then **4dHdrForeColor** will be used.

**4dHdrForeColor** — 1 to 256. The color at this position in 4D's palette will be used for the foreground color for the column header.

**alpListForeColor** — Name of the color in AreaList Pro's palette. This will be the foreground color for the column. If the name is not in AreaList Pro's palette or it is a null string, then **4dListForeColor** will be used.

**4dListForeColor** — 1 to 256. The color at this position in 4D's palette will be used for the foreground color for the column.

**alpFtrForeColor** — Name of the color in AreaList Pro's palette. If the name is not in AreaList Pro's palette or it is a null string, then **4dFtrForeColor** will be used.

**4dFtrForeColor** — 1 to 256. The color at this position in 4D's palette will be used for the foreground color for the column footer.

If **AL\_SetForeColor** is not called, the default is black for the header, list, and footer foreground colors.



## Configuring AreaList Pro Using Commands

**AL\_SetForeColor** can be used in the On load phase or in another phase (form event).

Examples:

`Red for column header foreground, light gray for column foreground (all columns),  
blue for footer foreground

**AL\_SetForeColor**(eList;0;"Red";0;"Light Gray";0;"Blue";0)

`Green for column header foreground, 13th color from 4D's palette for column foreground (4th column),  
7th color from 4D's palette for footer foreground

**AL\_SetForeColor**(eList;4;"Green";0;"";13;"";7)

---

### AL\_SetForeRGBColor

(areaRef:L; columnNumber:L; hdrForeRed:L; hdrForeGreen:L; hdrForeBlue:L; listForeRed:L;  
listForeGreen:L; listForeBlue:L; ftrForeRed:L; ftrForeGreen:L; ftrForeBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	longint	Column number
→ hdrForeRed	longint	Header foreground red
→ hdrForeGreen	longint	Header foreground green
→ hdrForeBlue	longint	Header foreground blue
→ listForeRed	longint	List foreground red
→ listForeGreen	longint	List foreground green
→ listForeBlue	longint	List foreground blue
→ ftrForeRed	longint	Footer foreground red
→ ftrForeGreen	longint	Footer foreground green
→ ftrForeBlue	longint	Footer foreground blue

**AL\_SetForeRGBColor** is used to specify the foreground colors for a column header, a list area column, and a column footer using the RGB values. This routine is similar to [AL\\_SetForeColor](#).

**hdrForeRed** — Desired header foreground red component in RGB color pattern.

**hdrForeGreen** — Desired header foreground green component in RGB color pattern.

**hdrForeBlue** — Desired header foreground blue component in RGB color pattern.

**listForeRed** — Desired list foreground red component in RGB color pattern.

**listForeGreen** — Desired list foreground green component in RGB color pattern.

**listForeBlue** — Desired list foreground blue component in RGB color pattern.

## Configuring AreaList Pro Using Commands

**ftrForeRed** — Desired footer foreground red component in RGB color pattern.

**ftrForeGreen** — Desired footer foreground green component in RGB color pattern.

**ftrForeBlue** — Desired footer foreground blue component in RGB color pattern.

The following example will tell AreaList Pro to draw the third column using a color scheme standard for OSX:

```
AL_SetForeRGBColor(eList;3;237;254;243;237;254;243;237;254;243)
```

---

### AL\_SetBackColor

(areaRef:L; columnNumber:I; alpHdrBackColor:S; 4dHdrBackColor:I; alpListBackColor:S; 4dListBackColor:I; alpFtrBackColor:S; 4dFtrBackColor:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column number
→ alpHdrBackColor	string	Header background color from Arealist Pro's palette
→ 4dHdrBackColor	integer	Header background color from 4D's palette
→ alpListBackColor	string	List background color from Arealist Pro's palette
→ 4dListBackColor	integer	List background color from 4D's palette
→ alpFtrBackColor	string	Footer background color from Arealist Pro's palette
→ 4dFtrBackColor	integer	Footer background color from 4D's palette

**AL\_SetBackColor** is used to specify the background colors for a column in the header, list area, and footer.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

**columnNumber** — The column for which to set the background color. Use a value of zero (0) for **columnNumber** to apply the parameters to all columns.

**alpHdrBackColor** — Name of the color in AreaList Pro's palette. This will be the background color for the column header. If the name is not in AreaList Pro's palette or it is a null string, then **4dHdrBackColor** will be used.

**4dHdrBackColor** — 1 to 256. The color at this position in 4D's palette will be used for the background color for the column header.

**alpListBackColor** — Name of the color in AreaList Pro's palette. This will be the background color for the column. If the name is not in AreaList Pro's palette or it is a null string, then **4dListBackColor** will be used.

## Configuring AreaList Pro Using Commands

**4dListBackColor** — 1 to 256. The color at this position in 4D's palette will be used for the background color for the column.

**alpFtrBackColor** — Name of the color in AreaList Pro's palette. This will be the background color for the column footer. If the name is not in AreaList Pro's palette or it is a null string, then **4dFtrBackColor** will be used.

**4dFtrBackColor** — 1 to 256. The color at this position in 4D's palette will be used for the background color for the column footer.

If **AL\_SetBackColor** is not called, the default is white for the header, list, and footer background colors.

**AL\_SetBackColor** can be used in the On load phase or in another phase (form event).

Examples:

```
`Light gray for header background, white for list background, all columns, gray for the footer background  
AL_SetBackColor(eList;0;"Light Gray";0;"White";0;"Gray";0)
```

```
`White for header background, 13th color from 4D's palette for list background, 1st column,  
color 246 from 4D's palette for footer background  
AL_SetBackColor(eList;1;"White";0;"";13;"";246)
```

---

## AL\_SetBackRGBColor

(areaRef:L; columnNumber:L; hdrBackRed:L; hdrBackGreen:L; hdrBackBlue:L; listBackRed:L; listBackGreen:L; listBackBlue:L; ftrBackRed:L; ftrBackGreen:L; ftrBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	longint	Column number
→ hdrBackRed	longint	Header back red
→ hdrBackGreen	longint	Header back green
→ hdrBackBlue	longint	Header back blue
→ listBackRed	longint	List back red
→ listBackGreen	longint	List back green
→ listBackBlue	longint	List back blue
→ ftrBackRed	longint	Footer back red
→ ftrBackGreen	longint	Footer back green
→ ftrBackBlue	longint	Footer back blue

## Configuring AreaList Pro Using Commands

**AL\_SetBackRGBColor** is used to specify the background colors for a column header, a list area column, and a column footer using the RGB values. This routine is similar to [AL\\_SetBackColor](#).

**hdrBackRed** — Desired header background red component in RGB color pattern.

**hdrBackGreen** — Desired header background green component in RGB color pattern.

**hdrBackBlue** — Desired header background blue component in RGB color pattern.

**listBackRed** — Desired list background red component in RGB color pattern.

**listBackGreen** — Desired list background green component in RGB color pattern.

**listBackBlue** — Desired list background blue component in RGB color pattern.

**fttBackRed** — Desired footer background red component in RGB color pattern.

**fttBackGreen** — Desired footer background green component in RGB color pattern.

**fttBackBlue** — Desired footer background blue component in RGB color pattern.

The following example will tell AreaList Pro to draw the third column using a color scheme standard for OSX:

```
AL_SetBackRGBColor(eList;3;237;254;243;237;254;243;237;254;243)
```

---

## AL\_SetDividers

(areaRef:L; colDividerPattern:S; alpColDividerColor:S; 4dColDividerColor:I;  
rowDividerPattern:S; alpRowDividerColor:S; 4dRowDividerColor:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ colDividerPattern	string	Pattern of the column divider
→ alpColDividerColor	string	Color from AreaList Pro's palette for the column divider
→ 4dColDividerColor	integer	Color from 4D's palette for the column divider
→ rowDividerPattern	string	Pattern of the row divider
→ alpRowDividerColor	string	Color from AreaList Pro's palette for the row divider
→ 4dRowDividerColor	integer	Color from 4D's palette for the row divider

**AL\_SetDividers** is used to set the pattern and color of the column and row dividers.

These are the available patterns: white, black, gray, light gray, and dark gray.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

## Configuring AreaList Pro Using Commands

**colDividerPattern** — Name of the pattern for the column divider. If a null string is used then no column divider will be displayed.

**alpColDividerColor** — Name of the color in AreaList Pro's palette. This will be the color for the column divider. If the name is not in AreaList Pro's palette or it is a null string, then **4dColDividerColor** will be used.

**4dColDividerColor** — 1 to 256. The color at this position in 4D's palette will be used for the column divider.

**rowDividerPattern** — Name of the pattern for the row divider. If a null string is used then no row divider will be displayed.

**alpRowDividerColor** — Name of the color in AreaList Pro's palette. This will be the color for the row divider. If the name is not in AreaList Pro's palette or it is a null string, then **4dRowDividerColor** will be used.

**4dRowDividerColor** — 1 to 256. The color at this position in 4D's palette will be used for the row divider.

If neither **AL\_SetDividers** nor [AL\\_SetRGBDividers](#) are called, then no column or row dividers will be displayed.

**AL\_SetDividers** can be used in the On load phase or in another phase (form event).

Examples:

‣ Display solid gray column dividers and no row dividers

**AL\_SetDividers**(eList;"Black";"Gray";0;"";"";0)

‣ Display column and row dividers in a gray pattern

**AL\_SetDividers**(eList;"Gray";"Black";0;"Gray";"Black";0)

### AL\_SetCellBorder

(areaRef:L; cellColumn:L; cellRow:L; borderLeft:L; borderTop:L; borderRight:L; borderBottom:L; offset:L; width:F; redColor:L; greenColor:L; blueColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellColumn	integer	Column
→ cellRow	longint	Row
→ borderLeft	integer	Draw left border
→ borderTop	integer	Draw top border
→ borderRight	integer	Draw right border
→ borderBottom	integer	Draw bottom border
→ offset	integer	Offset from cell boundary in pixels
→ width	real	Width of line
→ redColor	integer	Red
→ greenColor	integer	Green
→ blueColor	integer	Blue

**AL\_SetCellBorder** provides the ability to set the border style and RGB color for a cell.

**cellColumn** — Column of cell where border will be applied.

**cellRow** — Row of cell where border will be applied.

**borderLeft** — Draw left border.

**borderTop** — Draw top border.

**borderRight** — Draw right border.

**borderBottom** — Draw bottom border.

**offset** — Offset from cell boundary in pixels. 0 if the border should be drawn at cell boundary (default).

**width** — Width of line. Although this parameter is a real value, only integer widths will be drawn. Fractional widths (like 0.25 pixels) are used for compatibility with PrintList Pro's harline printing features.

**redColor** — RGB red component used for the border.

**greenColor** — RGB green component used for the border.

**blueColor** — RGB blue component used for the border.

### AL\_SetCellFrame

(areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; offset:I; width:F; redLightColor:I; greenLightColor:I; blueLightColor:I; redDarkColor:I; greenDarkColor:I; blueDarkColor:I; clearAllBorders:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ firstCellCol	integer	First cell column
→ firstCellRow	longint	First cell row
→ lastCellCol	integer	Last cell column
→ lastCellRow	longint	Last cell row
→ offset	integer	Offset from cell boundary in pixels
→ width	real	Width of line
→ redLightColor	integer	Red (light color)
→ greenLightColor	integer	Green (light color)
→ blueLightColor	integer	Blue (light color)
→ redDarkColor	integer	Red (dark color)
→ greenDarkColor	integer	Green (dark color)
→ blueDarkColor	integer	Blue (dark color)
→ clearAllBorders	integer	Clear all borders within the frame

**AL\_SetCellFrame** draws a frame around a range of cells. It uses RGB colors: light color for both left and top lines, dark color for both right and bottom line.

The range of cells from [firstCellCol, firstCellRow] to [lastCellCol, lastCellRow] will be set.

**offset** — Offset from cell boundaries in pixels. 0 if the frame should be drawn at cell boundaries (default).

**width** — Width of line. Although this parameter is a real value, only integer widths will be drawn. Future versions may allow fractional widths.

**redLightColor, greenLightColor, blueLightColor** — RGB components used for both left and top lines colors.

**redDarkColor, greenDarkColor, blueDarkColor** — RGB components used for both right and bottom lines colors.

**clearAllBorders** — If this parameter value is 1, then all cells inside the frame will have their borders removed.



### AL\_SetRGBDividers

(areaRef:L; colDividerPattern:S; colDividerRed:L; colDividerGreen:L; colDividerBlue:L; rowDividerPattern:S; rowDividerRed:L; rowDividerGreen:L; rowDividerBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ colDividerPattern	string	Column divider pattern string
→ colDividerRed	longint	Column divider — Red
→ colDividerGreen	longint	Column divider — Green
→ colDividerBlue	longint	Column divider — Blue
→ rowDividerPattern	string	Row divider pattern string
→ rowDividerRed	longint	Row divider — Red
→ rowDividerGreen	longint	Row divider — Green
→ rowDividerBlue	longint	Row divider — Blue

**AL\_SetRGBDividers** functions the same as the [AL\\_SetDividers](#) routine, except that the column and row divider colors use standard RGB values.

If neither **AL\_SetDividers** nor **AL\_SetRGBDividers** are called, then no column or row dividers will be displayed.

**colDividerPattern** — String, name of the pattern for the column divider. If a null string is used then no column divider will be displayed.

**colDividerRed** — Column divider RGB red component.

**colDividerGreen** — Column divider RGB green component.

**colDividerBlue** — Column divider RGB blue component.

**rowDividerPattern** — String, name of the pattern for the row divider. If a null string is used then no row divider will be displayed.

**rowDividerRed** — Row divider RGB red component.

**rowDividerGreen** — Row divider RGB green component.

**rowDividerBlue** — Row divider RGB blue component.

The following example will set the column/row dividers using the **AL\_SetRGBDividers** routine:

‘Display column and row dividers in a gray pattern

**AL\_SetRGBDividers**(eList;"Gray";209; 209; 209;"Gray"; 209; 209; 209)

### AL\_SetRowStyle

(areaRef:L; rowNumber:L; styleNum:I; fontName:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ rowNumber	longint	Number of row
→ styleNum	integer	Style of the font
→ fontName	string	Name of the font

**AL\_SetRowStyle** is used to set the type style and font for a particular row. It will override the style and font settings for all columns in that row. The size settings of each column will still apply.

**rowNumber** — The row for which to set the style. Use a value of zero (0) for **rowNumber** to apply the parameters to all rows.

**styleNum** — This parameter is used to set the style for the row. The different values in the table below can be added together to produce combinations of styles. For example, bold italic has a value of 3.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

If a row style has been previously set, it may be removed by setting **styleNum** to -1. This may also be applied to all rows by passing a zero (0) for the row number. This will have no effect on rows that have not been previously set.

The row style may be left unchanged by setting **styleNum** to 256.

**fontName** — This parameter specifies the font for a row. If a row font has been previously set, it may be removed by setting **fontName** to "1". Note that the value is a string, not a number. This may also be applied to all rows by passing a zero (0) for the row number. This will have no effect on rows that have not been previously set.

The row font may be left unchanged by setting **fontName** to the empty string ("").

See the **moveWithData** option of [AL\\_SetRowOpts](#). This controls whether row styles stay with their rows whenever sorting or dragging occurs.

## Configuring AreaList Pro Using Commands

Examples:

```
AL_SetRowStyle(eList;10;2;"") `set row 10 to be italic
```

```
AL_SetRowStyle(eList;0;1;"Helvetica") `set all rows to be bold, Helvetica
```

```
AL_SetRowStyle(eList;0;-1;"-1") `reset all row styles: column settings will be used
```

```
`Set the 12th row to display the Times font in bold italic style
```

```
AL_SetRowStyle(eList;12;3;"Times")
```

```
AL_UpdateArrays(eList;-1)
```

---

### AL\_SetRowColor

(areaRef:L; rowNumber:L; alpRowForeColor:S; 4dRowForeColor:L; alpRowBackColor:S; 4dRowBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ rowNumber	longint	Number of row
→ alpRowForeColor	string	Row foreground color from Arealist Pro's palette
→ 4dRowForeColor	longint	Row foreground color from 4D's palette
→ alpRowBackColor	string	Row background color from Arealist Pro's palette
→ 4dRowBackColor	longint	Row background color from 4D's palette

**AL\_SetRowColor** is used to specify the foreground and background colors for a row. It will override the foreground and background color settings for all columns in that row.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

**rowNumber** — The row for which to set the foreground color. Use a value of zero (0) for **rowNumber** to apply the parameters to all rows.

**alpRowForeColor** — Name of the color in AreaList Pro's palette. This will be the foreground color for the row. If the name is not in AreaList Pro's palette or it is a null string, then **4dRowForeColor** will be used.

**4dRowForeColor** — 1 to 256. Foreground color number for the row (from 4D's palette). If a row color has been previously set, it may be removed by setting **alpRowForeColor** to an empty string (""), and **4dRowForeColor** to -1. This may also be applied to all rows by passing a zero (0) for the **rowNumber**. This will have no effect on rows that have not been previously set.

The row foreground color may be left unchanged by setting **alpRowForeColor** to the empty string (""), and **4dRowForeColor** to 0.

**alpRowBackColor** — Name of the color in AreaList Pro's palette. This will be the background color for the row. If the name is not in AreaList Pro's palette or it is the empty string (""), then **4dRowBackColor** will be used.

## Configuring AreaList Pro Using Commands

**4dRowBackColor** —1 to 256. Background color number for the row (from 4D's palette).

If a row background color has been previously set, it may be removed by setting **alpRowBackColor** to the empty string (""), and **4dRowBackColor** to -1. This may also be applied to all rows by passing a zero (0) for the row number. This will have no effect on rows that have not been previously set.

The row background color may be left unchanged by setting **alpRowBackColor** to the empty string (""), and **4dRowBackColor** to 0.

See the **moveWithData** option of [AL\\_SetRowOpts](#). This controls whether row colors stay with their rows whenever sorting or dragging occurs.

Examples:

```
AL_SetRowColor(eList;10;"Blue";0;"Light gray";0) `set row 10 to foreground blue,  
background light gray
```

```
AL_SetRowColor(eList;0;"Blue";0;"Yellow";0) `set all rows to blue foreground, yellow background
```

```
AL_SetRowColor(eList;0;"";-1;"";-1) `reset all row colors to use the column color settings
```

```
AL_SetRowColor(eList;10;"Blue";0;"Light Gray";0) `set the 10th row to display a foreground color of  
blue and background color of light gray
```

```
AL_SetRowColor(eList;12;"Green";0;"";0) `set the 12th row to display a foreground color  
of green and the current background color
```

```
AL_UpdateArrays(eList;-1)
```

---

## AL\_SetRowRGBColor

(areaRef:L; rowNumber:L; rowForeRed:L; rowForeGreen:L; rowForeBlue:L; rowBackRed:L;  
rowBackGreen:L; rowBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ rowNumber	longint	Row number
→ rowForeRed	longint	Foreground red
→ rowForeGreen	longint	Foreground green
→ rowForeBlue	longint	Foreground blue
→ rowBackRed	longint	Background red
→ rowBackGreen	longint	Background green
→ rowBackBlue	longint	Background blue

**AL\_SetRowRGBColor** provides the ability to set the foreground and background colors for an individual row using standard RGB colors (see [AL\\_SetAltRowColor](#)).

This routine is similar to [AL\\_SetRowColor](#), except that it uses RGB color values.

## Configuring AreaList Pro Using Commands

**rowForeRed** — Desired foreground red component in RGB color pattern.

**rowForeGreen** — Desired foreground green component in RGB color pattern.

**rowForeBlue** — Desired foreground blue component in RGB color pattern.

**rowBackRed** — Desired background red component in RGB color pattern.

**rowBackGreen** — Desired background green component in RGB color pattern.

**rowBackBlue** — Desired background blue component in RGB color pattern.

The following example will tell AreaList Pro to draw the third row using a color scheme standard for OSX:

**AL\_SetRowRGBColor**(eList;3;237;0;243;0;254;0)

---

### AL\_SetAltRowColor

(areaRef:L; red:L; green:L; blue:L; options:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ red	longint	Red
→ green	longint	Green
→ blue	longint	Blue
→ options	longint	Options

**AL\_SetAltRowColor** provides the ability to set the alternate background row colors for an AreaList Pro area. The colors are defined using a standard RGB pattern and can optionally be configured to display the alternate row color in blank rows to fill the entire area with a consistent interface.

You may optionally display the alternate background row color for odd and/or even rows, including empty rows (those below the last row).

**red** — Desired red component in RGB color pattern.

**green** — Desired green component in RGB color pattern.

**blue** — Desired blue component in RGB color pattern.

**options** — Additionally formatting options (bitwise operator):

1 — display alternate background color in odd rows

2 — display alternate background color in even rows

## Configuring AreaList Pro Using Commands

The following example will tell AreaList Pro to draw the alternate rows using a color scheme standard for OSX:

```
AL_SetAltRowColor(eList;237;243;254;1)
```

This example will set the background color for odd rows to grey:

```
AL_SetAltRowColor(eList;209;209;209;1)
```

This example will set the background color for even rows to grey:

```
AL_SetAltRowColor(eList;209;209;209;2)
```

This example will set the background color for odd rows to white:

```
AL_SetAltRowColor (eList;255;255;255;1)
```

---

### AL\_SetAltRowClr

(areaRef:L; alpRowBackColor:S; 4dRowBackColor:I; options:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ alpRowBackColor	string	Alternate row background color from Arealist Pro's palette
→ 4dRowBackColor	integer	Alternate row background color from 4D's palette
→ options	longint	Options

**AL\_SetAltRowClr** performs the same action as [AL\\_SetAltRowColor](#), except it uses the standard AreaList Pro color formatting parameters as routines such as [AL\\_SetRowColor](#).

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

**alpRowBackColor** — Name of the color in AreaList Pro's palette. This will be the alternate row background color. If the name is not in AreaList Pro's palette or it is the empty string (""), then **4dRowBackColor** will be used.

**4dRowBackColor** — 1 to 256. Alternate row background color from 4D's palette.

**options** — Additionally formatting options (bitwise operator):

- 1 — display alternate background color in odd rows
- 2 — display alternate background color in even rows

## AL\_SetCellStyle

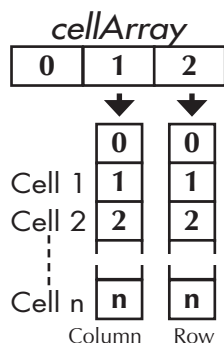
(areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; styleNum:I; fontName:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ firstCellCol	integer	First cell column
→ firstCellRow	longint	First cell row
→ lastCellCol	integer	Last cell column
→ lastCellRow	longint	Last cell row
→ cellArray	two-dimensional longint array	Discontiguous cells
→ styleNum	integer	Style of the font
→ fontName	string	Name of the font

**AL\_SetCellStyle** is used to set the font and/or style of a specific cell, range of cells, or list of cells.

- **To specify a single cell.** If **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** or **lastCellRow** are less than or equal to 0 then only [**firstCellCol**, **firstCellRow**] will be set.
- **To specify a range of cells.** If **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** and **lastCellRow** are greater than 0 then the range of cells from [**firstCellCol**, **firstCellRow**] to [**lastCellCol**, **lastCellRow**] will be set.
- **To specify discontiguous cells.** If **firstCellCol** or **firstCellRow** are less than or equal to 0 then the cells in **cellArray** will be set.

**cellArray** — Two-dimensional long integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



## Configuring AreaList Pro Using Commands

**styleNum** — This parameter is used to set the style for the specified cells. The values shown below can be added together to combine styles.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

If a cell style has been previously set, the style may be removed by setting **styleNum** to -1. The cell style may be left unchanged by setting **styleNum** to 256.

**fontName** — If a cell font has been previously set, it may be removed by setting **fontName** to "-1". Note that the value is a string, not a number. The cell font may be left unchanged by setting **fontName** to the empty string ("").

See the **moveWithData** option of [AL\\_SetCellsOpts](#). This controls whether cell styles and fonts stay with their cells whenever sorting, row dragging, or column dragging occurs.

Example:

Set the currently highlighted cell(s) to be bold

**ARRAY LONGINT** (aCellArray;2;0)

\$Result:=**AL\_GetCellSel**(eList;vCol1;vRow1;vCol2;vRow2;aCellArray)

**If** (\$Result=1)

**AL\_SetCellStyle**(eList;vCol1;vRow1;vCol2;vRow2;aCellArray;1;"")

**AL\_UpdateArrays**(eList;-1)

**End if**



### AL\_GetCellStyle

(areaRef:L; cellColumn:I; cellRow:L; styleNum:I; fontName:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellColumn	integer	Cell column
→ cellRow	longint	Cell row
← styleNum	integer	Style of the font
← fontName	string	Name of the font

**AL\_GetCellStyle** is used to get the font and/or style of a particular cell. It will not get the column or row font and/or style.

**cellColumn** — Column of cell where to get the style.

**cellRow** — Row of cell where to get the style.

**styleNum** — This parameter returns the style number for the cell. The number can be a sum of several individual styles. For example, bold italic has a value of 3.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16
Condensed	32
Extended	64

If a cell style has not been previously set, the value of **styleNum** will be -1.

**fontName** — If a cell font has not been previously set, the value of **fontName** will be “-1”. Note that the value is a string, not a number.

Example:

```
`Get the style of the cell in the third column, first row  
AL_GetCellStyle(eList;3;1;vStyle;vFont)
```

## AL\_SetCellColor

(areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X;  
alpForeColor:S; 4dForeColor:I; alpBackColor:S; 4dBackColor:I)

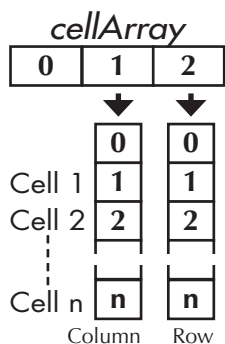
Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ firstCellCol	integer	First cell column
→ firstCellRow	longint	First cell row
→ lastCellCol	integer	Last cell column
→ lastCellRow	longint	Last cell row
→ cellArray	two-dimensional longint array	Discontiguous cells
→ alpForeColor	string	Foreground color from AreaList Pro's palette
→ 4dForeColor	integer	Foreground color from 4D's palette
→ alpBackColor	string	Background color from AreaList Pro's palette
→ 4dBackColor	integer	Background color from 4D's palette

**AL\_SetCellColor** is used to set the foreground color and/or background color of a specific cell, range of cells, or list of cells.

- **To specify a single cell.** If **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** or **lastCellRow** are less than or equal to 0 then only [firstCellCol, firstCellRow] will be set.
- **To specify a range of cells.** If **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** and **lastCellRow** are greater than 0 then the range of cells from [firstCellCol, firstCellRow] to [lastCellCol, lastCellRow] will be set.
- **To specify discontiguous cells.** If **firstCellCol** or **firstCellRow** are less than or equal to 0 then the cells in **cellArray** will be set.

AreaList Pro has its own palette, with the following colors: white, black, blue, green, yellow, magenta, red, cyan, gray, light gray.

**cellArray** — Two-dimensional long integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



## Configuring AreaList Pro Using Commands

**alpForeColor** — Name of the color in AreaList Pro's palette. This will be the foreground color for the cell. If the name is not in AreaList Pro's palette or it is the empty string (""), then **4dForeColor** will be used.

**4dForeColor** — 1 to 256. Foreground color number for the cell (from 4D's palette). If a cell foreground color has been previously set, it may be removed by setting **alpForeColor** to the empty string (""), and **4dForeColor** to 1. The cell foreground color may be left unchanged by setting **alpForeColor** to the empty string (""), and **4dForeColor** to 0.

**alpBackColor** — Name of the color in AreaList Pro's palette. This will be the background color for the cell. If the name is not in AreaList Pro's palette or it is the empty string (""), then **4dBackColor** will be used.

**4dBackColor** — 1 to 256. Background color number for the cell (from 4D's palette). If a cell background color has been previously set, it may be removed by setting **alpBackColor** to the empty string (""), and **4dBackColor** to 1. The cell background color may be left unchanged by setting **alpBackColor** to the empty string (""), and **4dBackColor** to 0.

The foreground and background colors for a cell may be set differently during data entry by calling **AL\_SetCellColor** in the entry started callback method and again in the entry finished callback method to restore the colors.

See the **moveWithData** option of [AL\\_SetCellsOpts](#). This controls whether cell foreground and background colors stay with their cells whenever sorting, row dragging, or column dragging occurs.

Example:

```
`Set all negative values in the third column, a real array, to have a foreground color of red
ARRAY LONGINT (aCellArray;2;0) `MUST initialize a two-dimensional long integer array
For ($i;1;Size of array (aRevenue)) `check each element in the array
  If (aRevenue{$i}<0) `is the value in this element negative?
    AL_SetCellColor (eList;3;$i;0;0;aCellArray;"Red";0;"";0) `if so, then show it in red
  End if
End for
AL_UpdateArrays (eList;-1)
```

### AL\_GetCellColor

(areaRef:L; cellColumn:L; cellRow:L; 4dForeColor:L; 4dBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellColumn	integer	Cell column
→ cellRow	longint	Cell row
← 4dForeColor	integer	Foreground color from 4D's palette
← 4dBackColor	integer	Background color from 4D's palette

**AL\_GetCellColor** is used to get the foreground color and/or background color of a specific cell. It will not get the column or row foreground color and/or background color.

For this command to function correctly the cell foreground and background colors must have been set from 4D's palette. In other words, the **4dForeColor** and **4dBackColor** parameters must have been used in the command [AL\\_SetCellColor](#).

**cellColumn** — Column of cell where to get the color.

**cellRow** — Row of cell where to get the color.

**4dForeColor** — 1 to 256. Foreground color number for the cell (from 4D's palette).  
If a cell foreground color has not been previously set, the value of **4dForeColor** will be -1.

**4dBackColor** — 1 to 256. Background color number for the cell (from 4D's palette).  
If a cell background color has not been previously set, the value of **4dBackColor** will be -1.

### AL\_SetCellRGBColor

(areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X;  
cellForeRed:L; cellForeGreen:L; cellForeBlue:L; cellBackRed:L; cellBackGreen:L;  
cellBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ firstCellCol	integer	First cell column
→ firstCellRow	longint	First cell row
→ lastCellCol	integer	Last cell column
→ lastCellRow	longint	Last cell row
→ cellArray	two-dimensional longint array	Discontiguous cells
→ cellForeRed	longint	Foreground red
→ cellForeGreen	longint	Foreground green
→ cellForeBlue	longint	Foreground blue
→ cellBackRed	longint	Background red
→ cellBackGreen	longint	Background green
→ cellBackBlue	longint	Background blue

**AL\_SetCellRGBColor** is used to set the foreground and/or background color of a specific cell, range of cells, or list of cells. This routine works in the same manner as [AL\\_SetCellColor](#), except it allows you to specify the colors using standard RGB values.

**cellForeRed** — Desired foreground red component in RGB color pattern.

**cellForeGreen** — Desired foreground green component in RGB color pattern.

**cellForeBlue** — Desired foreground blue component in RGB color pattern.

**cellBackRed** — Desired background red component in RGB color pattern.

**cellBackGreen** — Desired background green component in RGB color pattern.

**cellBackBlue** — Desired background blue component in RGB color pattern.

### AL\_GetCellRGBColor

(areaRef:L; cellColumn:I; cellRow:L; cellForeRed:L; cellForeGreen:L; cellForeBlue:L; cellBackRed:L; cellBackGreen:L; cellBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellColumn	integer	Cell column
→ cellRow	longint	Cell row
← cellForeRed	longint	Foreground red
← cellForeGreen	longint	Foreground green
← cellForeBlue	longint	Foreground blue
← cellBackRed	longint	Background red
← cellBackGreen	longint	Background green
← cellBackBlue	longint	Background blue

**AL\_GetCellRGBColor** is used to get the foreground and/or background color of a specific cell.

This routine works in the same manner as [AL\\_GetCellColor](#), except it allows you to get the color information using standard RGB values.

See [AL\\_SetCellRGBColor](#) for details about the parameters.

### AL\_SetCellSel

(areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ firstCellCol	integer	First cell column
→ firstCellRow	longint	First cell row
→ lastCellCol	integer	Last cell column
→ lastCellRow	longint	Last cell row
→ cellArray	two-dimensional longint array	Discontiguous cells

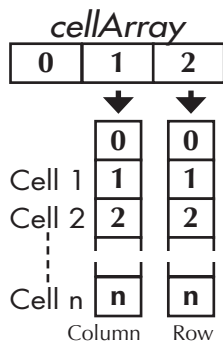
**AL\_SetCellSel** is used to set the cell selection.

Use the **cellSelection** option of [AL\\_SetCellOpts](#) to specify a cell selection mode prior to using this command.

## Configuring AreaList Pro Using Commands

- **To select a single cell.** If `firstCellCol` and `firstCellRow` are greater than 0 and `lastCellCol` or `lastCellRow` are less than or equal to 0 then only `[firstCellCol, firstCellRow]` will be selected.
- **To select a range of cells.** If `firstCellCol` and `firstCellRow` are greater than 0 and `lastCellCol` and `lastCellRow` are greater than 0 then the range of cells from `[firstCellCol, firstCellRow]` to `[lastCellCol, lastCellRow]` will be selected.
- **To select discontinuous cells.** If `firstCellCol` or `firstCellRow` are less than or equal to 0 then the cells in `cellArray` will be selected.

**cellArray** — Two-dimensional long integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



Examples:

**AL\_SetCellSel**(eList;1;3;0;0) `select cell at column 1, row 3

**AL\_SetCellSel**(eList;2;2;5;5) `select cells from column 2, row 2 to column 5, row 5

**ARRAY LONGINT**(aCellSelect;2;4)

aCellSelect{1}{1}:=1 `column 1

aCellSelect{2}{1}:=1 `row 1

aCellSelect{1}{2}:=1 `column 1

aCellSelect{2}{2}:=2 `row 2

aCellSelect{1}{3}:=2 `column 2

aCellSelect{2}{3}:=5 `row 5

aCellSelect{1}{4}:=2 `column 2

aCellSelect{2}{4}:=6 `row 6

**AL\_SetCellSel**(eList;0;0;0;0;aCellSelect) `select the cells in aCellSelect

[AL\\_GetCellSel](#) is used to get the cell selection.

### AL\_SetSort

(areaRef:L; column1:I; ...; columnN:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ column1; ...; columnN	integer	Column(s) to perform sort upon

**AL\_SetSort** is used to perform a multi-level sort.

**column** — These parameters specify the columns to use for the sort criteria.

- A **column** greater than 0 causes an ascending sort to be performed upon that column, while a **column** less than 0 causes a descending sort to be performed upon that column. The arrow indicator will be up or down accordingly for the column. If a **column** is 0, or it is a picture array or field, or it contains a field from a related one table, then all subsequent columns will be ignored.
- If the first **column** has a value other than 0, then the sort indicator will be displayed in its header accordingly. If the first **column** has a value of 0, then AreaList Pro will not sort the columns and no sort indicator will be displayed.
- If the first two **column** parameters have the same value, then AreaList Pro will not sort the columns, but the sort indicator will be displayed in its header accordingly for the first **column**.

You can determine what columns a user has sorted using [AL\\_GetSort](#).

Examples:

**AL\_SetSort**(eList;3;4;7) `sort on columns 3, 4, and 7 (all ascending)

**AL\_SetSort**(eList;-1;3;-2) `sort on columns 1 (descending), 3 (ascending), and 2 (descending)

**AL\_SetSort**(eList;0) `don't sort, and don't display any sort indicator

**AL\_SetSort**(eList;2;2) `don't sort, but do display sort indicator in the header for column 2



### AL\_SetCellValue

(areaRef:L; row:L; column:I; alphaNumericData:S; pictData:P)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellRow	longint	Row number
→ cellColumn	integer	Column number
→ alphaNumericData	string	Alphanumeric value
→ pictData	picture	Picture data

**AL\_SetCellValue** provides the ability to update the contents of a given cell. You can set either alphanumeric or picture data.

**cellRow** — Cell row number.

**cellColumn** — Cell column number.

**alphaNumericData** — Alphanumeric (non-picture) data you wish to use as new value.

**pictData** — Picture data you wish to use as new value.

See [AL\\_GetClickedRow](#) for an example of using **AL\_SetCellValue**.

### AL\_SetLine

(areaRef:L; rowNumber:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ rowNumber	longint	Row number to select (highlight)

**AL\_SetLine** is used to set the row to be highlighted. This command is used in the On load phase to set up the initial display of an AreaList Pro object. You can also use it in other phases to control what element is selected. If this command is not used, then AreaList Pro will display the columns with the first row selected.

## Configuring AreaList Pro Using Commands

**AL\_SetLine** should only be used with an AreaList Pro object in single-row mode. If **areaRef** is in multi-rows mode, you must use [AL\\_SetSelect](#).

**rowNumber** — This parameter specifies what row to highlight.

Example:

### Case of

: (**Form event**=On Load)

\$error:=**AL\_SetArraysNam**(eList;1;3;"aFN";"aLN";"aComp")

**AL\_SetLine**(eList;3) `highlight 3rd row

### End case

---

## AL\_SetSelect

(areaRef:L; rowsToSelect:X)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ rowsToSelect	longint array	Contains element numbers to select (highlight) when the multi-rows option is enabled

**AL\_SetSelect** is used to set the rows to be highlighted. This command is used in the On load phase to set up the initial display of an AreaList Pro object. You can also use it in other phases to control what elements are selected. If this command is not used, then AreaList Pro will display the columns with no rows selected.

**AL\_SetSelect** should only be used with an AreaList Pro object in multi-rows mode. If **areaRef** is in single-row mode, you must use [AL\\_SetLine](#).

**rowsToSelect** — Long integer array. This parameter contains a list of rows which you wish to select, or highlight.

Example:

`eList AreaList Pro object method

### Case of

: (**Form event**=On Load)

**ARRAY LONGINT**(aRows;2) `create an long integer array with 2 elements

aRows{1}:=1 `set row 1 to be highlighted

aRows{2}:=3 `and row 3 to be highlighted

\$error:=**AL\_SetArraysNam**(eList;1;2;"aFN";"aLN") `specify arrays to display

**AL\_SetSelect**(eList;aRows) `specify the rows to highlight

### End case

### AL\_SetScroll

(areaRef:L; verticalScroll:L; horizontalScroll:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ verticalScroll	longint	Vertical position (element #) to scroll to
→ horizontalScroll	integer	Horizontal position (column #) to scroll to

**AL\_SetScroll** is used to set the position of the thumb in the vertical and horizontal scrollbars.

**verticalScroll** — This parameter represents the element number to display at the top of the AreaList Pro display.

**horizontalScroll** — This parameter represents the column number to display at the left of the AreaList Pro display.

The value passed to **horizontalScroll** represents the actual column number, including any columns which might be currently locked. For example, if the two left columns are locked, and you want to scroll the list one column to the left, so that the fourth column is adjacent to the second locked column, then the value to pass is four.

**AL\_SetScroll** can also be used to hide or show the vertical and horizontal scrollbars. The possible values to use to hide or show the scrollbars are shown in the table below. The default is that both scrollbars are shown.

Value	VertScroll	HorizScroll
>0	Vertical scroll position	Horizontal scroll position
0	Hide when displaying another form (required)	Hide when displaying another form (required)
-1	Hide if shown, Show if hidden	Hide if shown, Show if hidden
-2	Show	Show
-3	Hide	Hide

When using **AL\_SetScroll** to hide or show the scrollbars, either [AL\\_UpdateArrays](#) with `updateMethod` set to -2, or [AL\\_UpdateFields](#) with `updateMethod` set to 2 must be called.

**AL\_SetScroll** can still be used to set the scroll position even with the scrollbar(s) hidden.

AreaList Pro automatically hides the horizontal scrollbar if `allowColumnResize` in [AL\\_SetColOpts](#) is set to 0 and all of the displayed columns fit within the width of the list area. AreaList Pro automatically shows the horizontal scrollbar if `allowColumnResize` in **AL\_SetColOpts** is set to 1 or all of the displayed columns do not fit within the width of the list area.

## Configuring AreaList Pro Using Commands

If the horizontal scrollbar is shown or hidden manually by passing -1, -2 or -3 in the `horizontalScroll` parameter of **AL\_SetScroll**, then this behavior will be permanently disabled for the AreaList Pro object.

---

*Pass values of zero for `verticalScroll` and `horizontalScroll` if another form is going to be displayed in the window with **DIALOG**, **ADD RECORD** or **MODIFY RECORD** commands. This is required to inform the AreaList Pro object that another form will be displayed. Neither **AL\_UpdateArrays** nor **AL\_UpdateFields** should be called in this specific case. See [Scroll bars — Changing Displayed Form](#).*

---

**AL\_SetScroll** can be used in the On load phase or in another phase (form event).

Examples:

Set an AreaList Pro object to display the 15th element

the object is named eList

**AL\_SetScroll**(eList;15;1)

Configure the AreaList Pro object not to display the vertical scrollbar

If (Form event=On Load) `do any desired setup, then hide the vertical scrollbar

**AL\_SetScroll**(eList;-1;1)

End if

---

## AL\_SetColLock

(areaRef:L; columns:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columns	integer	Number of columns to lock

**AL\_SetColLock** is used to set the number of columns to lock. AreaList Pro will not allow more columns to be locked than the number of displayed columns minus two.

**columns** — This parameter is used to specify the number of columns to lock.

**AL\_SetColLock** can be used in the On load phase or in another phase (form event).

Example:

**AL\_SetColLock**(eList;2) `lock the first two columns

### AL\_SetHeight

(areaRef:L; numHeaderLines:I; headerHeightPad:I; numRowsLines:I; rowHeightPad:I; numFooterLines:I; footerHeightPad:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ numHeaderLines	integer	Number of text lines in the header
→ headerHeightPad	integer	Extra height for the header
→ numRowsLines	integer	Number of text lines in each row
→ rowHeightPad	integer	Extra height for each row
→ numFooterLines	integer	Number of text lines in the footer
→ footerHeightPad	integer	Extra height for the footer

**AL\_SetHeight** is used to set the number of lines of text along with additional height padding in the header, in the rows, and in the footer. Only text and string columns can wrap to more than one line.

If **numRowLines** is set to 2 or more, text and string elements will be able to wrap into the number of lines specified for each row. Note that all rows will be given the same number of lines regardless of the actual number of lines used by a specific text or string element.

Additional padding may be set using **rowHeightPad** to allow more space between rows. Text will be centered vertically in the header or row. Note that the padding applies to the entire row and not on a line by line basis within the row.

**numHeaderLines** — The number of lines in the header. Default is 1.

**headerHeightPad** — The extra height, in pixels, to give to the header. Default is 2.

**numRowLines** — The number of lines to give to each row. Default is 1.

**rowHeightPad** — The extra height, in pixels, to give to each row. Default is 0.

**numFooterLines** — The number of lines to give to the footer. Default is 1.

**footerHeightPad** — The extra height, in pixels, to give to the footer. Default is 2.

Examples:

**AL\_SetHeight**(elist;1;4;1;2;1;4) `pad the header by 4 pixels, the rows by 2, the footers by 4

**AL\_SetHeight**(elist;2;5;2;0;2;0) `set header lines to 2, pad to 5 pixels, set row lines to 2, no padding, set footer lines to 2, no padding

### AL\_SetMinRowHeight

(areaRef:L; minRowHeight:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ minRowHeight	longint	Minimum row height

**AL\_SetMinRowHeight** provides the ability to set the minimum row height for AreaList Pro rows. This is different than row padding as it will allow you to set individual rows to appear with extra white space, regardless of the amount of data.

**minRowHeight** — Minimum row height.

The following example will set the minimum row height to two rows, regardless of the amount of data displayed:

```
AL_SetMinRowHeight(eList;2)
```

### AL\_SetPictureEscape

(areaRef:L; escapeChar:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ escapeChar	string	Escape character

**AL\_SetPictureEscape** will set the current escape character used to inform AreaList Pro where icon references exist in your cell data or headers.

You have the ability to include icons within AreaList Pro headers and cell data using a formatted character (default ^) to informing AreaList Pro where to look for the icons.

For more details on using header and cell data icons, please refer to the [Header/Cell Icon Support](#) section.

**escapeChar** — Sets the alternate escape character.

The following example will display the “cicn” resource with a **resID** of 150 in the header before the header text:

```
AL_SetPictureEscape(area;"~") `set escape to tilde ~  
AL_SetHeaders(area;1;1;"~150Header")
```

### AL\_GetPictureEscape

(areaRef:L) → escapeChar:S

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← escapeChar	string	Escape character

**AL\_GetPictureEscape** will return the current escape character used to inform AreaList Pro where icon references exist in your cell data or headers.

**escapeChar** — Returns the alternate escape character.

The following example will return the escape character we just set in [AL\\_SetPictureEscape](#):

```
AL_SetPictureEscape(area;"~") `set escape to tilde ~  
$char:=AL_GetPictureEscape(area) `returns "~" in $char
```

# Using the Callback Methods

A “callback” is a 4D project method which is executed by a plug-in. AreaList Pro lets you make use of callbacks when entering and exiting an AreaList Pro object. This feature provides you with the ability to enable/disable buttons or other variables depending upon which object is active.

See [Executing a Callback Upon Entering an Area](#) and [Executing a Callback Upon Exiting an Area](#).

See [Redrawing the Display from the Callback Method](#) for more information on updating buttons or other variables from a callback method.

You can also use callback methods when entering and exiting a cell during data entry. Please read the section [Using Callback Methods During Data Entry](#) for more information.

The [Event Callback Interface](#) is an alternate method for responding to AreaList Pro events, the other being the [On Plug in Area/AL\\_GetLastEvent](#) (formerly **ALProEvt** variable) combination (see [User Action Commands](#) and [Event Callback vs Object Method](#)).

The [Edit Menu Callback](#) system provides an interface of overriding the default behavior when working with the 4D edit menu.

In addition, a callback is available when the user clicks on the icon which is displayed over the scrollbars (sort area). See [AL\\_SetHeaderOptions](#).

## Summary

AreaList Pro provides eight different callback methods:

- entering an AreaList Pro area (**areaEnteredMethod** parameter of [AL\\_SetMainCalls](#))
- exiting an AreaList Pro area (**areaExitedMethod** parameter of [AL\\_SetMainCalls](#))
- entering a cell (**entryStartedMethod** parameter of [AL\\_SetCallbacks](#))
- exiting a cell (**entryFinishedMethod** parameter of [AL\\_SetCallbacks](#))
- events occurring while in an area (**callbackMethod** parameter of [AL\\_SetEventCallback](#))
- actions on the Edit menu while in an AreaList Pro area (**callbackMethod** parameter of [AL\\_SetEditMenuCallback](#))
- calculated columns (**calcCallback** parameter of [AL\\_SetCalcCall](#))
- clic on a column header (**callbackMethod** parameter of [AL\\_SetHeaderOptions](#))



### Warnings

- Callback methods called during cell editing (cell enter, cell exit and edit menu) must not modify underlying data (arrays or records) i.e. must not resize or rebuild the arrays (array display) or change the current 4D selection (field display).
- [AL\\_UpdateArrays](#) can only be called with `updateMethod` equal to -1 from a callback method **other than an event callback**. Please read the section [Modifying Array Elements Procedurally](#) for more information.  
[AL\\_UpdateFields](#) can only be called with `updateMethod` equal to 0 or 1 from a callback method **other than an event callback**. The call will be ignored if parameter -2 is used **in other callbacks**.
- You should not call any AreaList Pro commands which change the number of displayed columns, their position in the area, or their sorted order.
- All callbacks receive the area long integer reference as their first parameter (\$1). You must use the following declaration in your callback method:

**C\_LONGINT(\$1)**

Since the long integer \$1 parameter contains 4D's representation of the AreaList Pro object, it can be used as the first parameter of any AreaList Pro method called.

In addition, some callback methods receive other parameters, which need to be declared as well as documented below.

---

***Callback methods must not add or delete any columns.***

---

### Executing a Callback Upon Entering an Area

An "area entered" callback method is a 4<sup>th</sup> Dimension project method called whenever the AreaList Pro object is entered. The area entered callback method is specified by passing the method name in the `areaEnteredMethod` parameter of [AL\\_SetMainCalls](#). If this parameter is a null string then no method will be called.

The area entered callback method is passed one parameter by AreaList Pro. This parameter is a long integer that corresponds to the AreaList Pro object on the layout.

You must use the following declaration in your callback method:

**C\_LONGINT(\$1)**

You can call [AL\\_GotoCell](#) from the area entered callback to initiate data entry when the object is entered.

### Executing a Callback Upon Exiting an Area

An “area exited” callback method is a 4<sup>th</sup> Dimension project method called whenever the AreaList Pro object is exited. The area exited callback method is specified by passing the method name in the `areaExitedMethod` parameter of [AL\\_SetMainCalls](#). If this parameter is a null string then no method will be called.

The area exited callback method is passed one parameter by AreaList Pro. This parameter is a long integer that corresponds to the AreaList Pro object on the layout.

You must use the following declaration in your callback method:

**C\_LONGINT(\$1)**

### Using Callback Methods During Data Entry

AreaList Pro lets you make use of callbacks when entering and exiting a cell, and when a popup menu is clicked or released. This feature provides you with considerable control over user actions, allowing you to do such things as reject an entry, provide a choice list, or simply skip a particular cell.

[AL\\_UpdateArrays](#) can only be called with `updateMethod` equal to -1 and [AL\\_UpdateFields](#) can only be called with `updateMethod` equal to 0 or 1 from a callback method **other than an event callback**.

In addition to altering the array content, you can change color and style, reject or accept entered data, and change the current data entry cell using the AreaList Pro commands listed above. You should not call any command which changes the number of displayed arrays, their position in the area, or their sorted order. See [Event Callback vs Object Method](#).

### Executing a Callback Upon Entering a Cell

An “entry started” callback method is a 4<sup>th</sup> Dimension method called when data entry begins for a cell or an AreaList Pro popup menu is clicked, and is specified by passing the method name in the `entryStartedMethod` parameter of [AL\\_SetCallbacks](#). If this parameter is a null string then no method will be called.

AreaList Pro will pass the `entryStartedMethod` callback method two parameters if arrays are being displayed, or three parameters if fields are displayed.

- the first parameter is a long integer that corresponds to the AreaList Pro object on the layout
- the second parameter is a long integer that reports what action (mode) caused data entry to begin in the cell
- the third parameter is a long integer that reports whether the record was loaded or not (when fields are being displayed)

You must use the following declaration in your callback method:

**C\_LONGINT(\$1;\$2;\$3)**

## Using the Callback Methods

As stated above, the second parameter passed to the callback routine, the long integer \$2, contains the mode by which data entry began, according to the following table:

Constant	Value	Entry Mode
AL Click action	1	Click in Cell
AL Tab key action	2	Tab
AL Shift_Tab key action	3	Shift-Tab
AL Return key action	4	Return
AL Shift_Return key action	5	Shift-Return
AL GotoCell action	6	<a href="#">AL_GotoCell</a>
	7	Not used
	8	Not used
AL SkipCell action	9	<a href="#">AL_SkipCell</a>
AL Other cell popup action	10	Click on cell popup when cursor not already in cell
AL Active cell popup action	11	Click on cell popup when cursor already in cell

The **entryStartedMethod** callback is also executed whenever a popup menu is clicked, but before the menu is actually displayed. When this occurs, the **entryStartedMode** (\$2) provided by AreaList Pro will be 10 if the popup was clicked on a cell other than the one actively in data entry. Mode 11 will be reported if data entry was already established in the cell for which the popup was clicked.

One of the primary uses of the **entryStartedMethod** callback when the popup is clicked would be to load the array from which the popup is built, then use [AL\\_SetEnterable](#) to pass the array to AreaList Pro.

If the third parameter is 1, then the record was loaded properly and the field contents can be edited. If the third parameter is 0, then the record is locked by another process or user.

If typed data entry is underway and the record can not be loaded, then [AL\\_GotoCell](#) or [AL\\_SkipCell](#) may be used to continue data entry in another cell.

If neither of these commands is called then data entry will end. If popup data entry is underway and the record can not be loaded then data entry will end.

## Executing a Callback Upon Leaving a Cell

An “entry finished” callback method is a 4<sup>th</sup> Dimension project method called when data entry ends for a cell, or when an AreaList Pro popup menu is released for a cell not in typed data entry. The entry finished callback method is specified by passing the method name in the **entryFinishedMethod** parameter of [AL\\_SetCallbacks](#). If this parameter is a null string then no method will be called.

The **entryFinishedMethod** callback method is passed two parameters by AreaList Pro. The first parameter is a long integer that corresponds to the AreaList Pro object on the layout. The second parameter is a long integer that reports what action (mode) caused data entry to end in the cell.

You must use the following declarations in your entry finished callback method:

**C\_BOOLEAN** (\$0) `allow cell exit

**C\_LONGINT** (\$1;\$2)

## Using the Callback Methods

As stated above, the second parameter passed to the callback routine, the long integer \$2, contains the mode by which data entry ended, according to the following table:

Constant	Value	Exit Mode
AL Click action	1	Click outside cell on object
AL Tab key action	2	Tab
AL Shift_Tab key action	3	Shift-Tab
AL Return key action	4	Return
AL Shift_Return key action	5	Shift-Return
AL GotoCell action	6	<a href="#">AL_GotoCell</a>
AL ExitCell action	7	<a href="#">AL_ExitCell</a> or “hard deselect”
AL Cell validate action	8	Deselect the cell (“soft deselect”)
	9	Not used
AL Other cell popup action	10	Cell popup released when cursor not already in cell
AL Active cell popup action	11	Cell popup released when cursor already in cell

See the sections [Compatibility Note — New Menu Architecture](#) and [Compatibility Note — AL ExitCell and AL Cell deselect action become AL ExitCell and AL Cell Validate](#) below for details about “hard deselect” (mode 7) and “soft deselect” (mode 8)

The `entryFinishedMethod` callback method is actually a function. It must return **True** for the value entered into the cell to be accepted, and **False** for the value to be rejected. If the value is rejected the user will not be allowed to leave the cell.

See the 4<sup>th</sup> Dimension Language Reference for more details about functions and methods.

The `entryFinishedMethod` callback function is also called when a popup menu is released. In this case, the `entryFinishedMode` (\$2) reported by AreaList Pro to the callback will be 10 if typed data entry was in progress in the cell which contains the popup, or 11 if typed data entry was not in progress in that cell.

[AL\\_GotoCell](#) can be used to establish typed data entry on the cell if it did not exist before the popup was clicked.

---

***If typed data entry is already established for the cell in which the popup exists, the entry finished callback function will not run when the popup menu is released.***

---

When displaying arrays and data entry is initiated in a cell, the contents of the array element will be copied into the zero element of the array being displayed in the column. Please read the section [Initiating Data Entry](#) for more information.

When fields are displayed, the contents of the field are not copied. Thus it is up to you to save the field contents in the entry started callback method if they will be needed for comparison in the entry finished callback method.

When displaying arrays and the entry finished callback method is executed, the array element corresponding to the cell has already been updated with the new value that was entered by the user.

## Using the Callback Methods

Thus, the zero element which contains the old data and the element representing the current cell can both be used to determine data validity.

Among the possible situations and responses that may occur are the following:

- The data is valid. Set `$0:=True` to complete data entry for the cell.
- The data is invalid. Copy the old data from the zero element to the array element corresponding to the cell. Set `$0:=True` to complete data entry for the cell.

For example:

```
aFname{vRow}:=aFname{0} `reset the cell contents to their original state
```

```
$0:=True
```

- The data is invalid. Inform the user that the data is invalid. Set `$0:=False` to force the user to remain in the cell and enter another value.
- The data is invalid. Inform the user that the data is invalid. Modify the cell contents, call [AL\\_GotoCell](#) to go to the current cell, and set `$0:=True`. This achieves the same effect as rejecting the entry, but allows the cell contents to be modified.

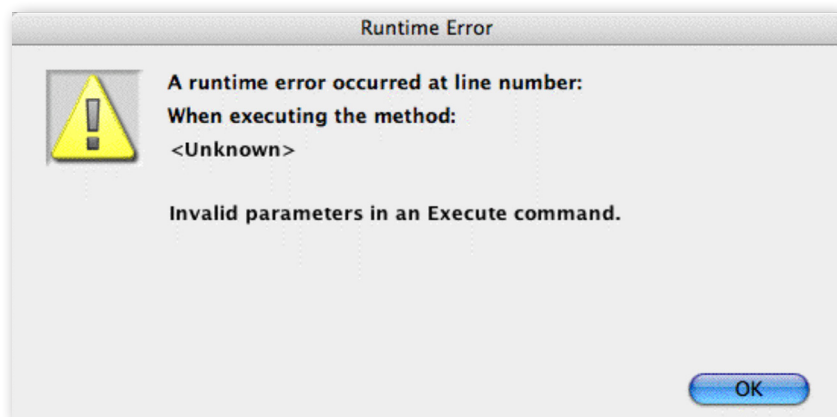
For example:

```
aFname{vRow}:=aFname{0} `reset the cell contents to their original state
```

```
AL_GotoCell(eList;vColumn;vRow) `go to the same cell
```

```
$0:=True
```

When using an exit callback method, AreaList Pro requires the existence of a result value (**C\_BOOLEAN**) and failure to have the callback method incorrectly declared will produce an error in compiled applications:



COMPILED RUNTIME ERROR WHEN \$0 NOT DECLARED

Your callback method should be declared as:

```
C_BOOLEAN($0) `required
```

```
C_LONGINT($1)
```

```
C_LONGINT($2)
```

### Compatibility Note — New Menu Architecture

Since version 7.9, it is no longer required to customize the cell exit callback when enterability is active with the “New Menu Architecture” active. You no longer need to trap and conditionally respond to [AL Cell validate action](#) (mode 8).

AreaList Pro handles correctly Edit menu events if the new Edit menu behavior is set. However, this may break compatibility with existing code. If the number 8 is added to the first parameter of [AL SetEntryOpts](#) call, AreaList Pro will switch to its previous behavior.

The difference between the two behaviors is as follows: 4D sends two kinds of deselect events to the plug-in area, real (hard) deselect and validating (soft) deselect.

- The first mode ([AL ExitCell action](#)) means that the user clicked on another focusable object (like an edit field) and the focus is going to pass from the AreaList Pro area to the new object.
- The second mode ([AL Cell validate action](#)) means that the user clicked on some non-focusable object and the focus will stay on the AreaList Pro area.

However, the difference between focusable and non-focusable objects is not obvious for users, so AreaList Pro until version 7.8 did not handle the two events differently. With versions 7.9 and above, it has to, as 4D sends the soft deselect to AreaList Pro before it sends edit menu commands.

Soft deselect is passed to the exit callback method with the `entryFinishedMode` parameter set to the value 8 ([AL Cell validate action](#)).

When the cell exit callback method is called with this value, it means that editing will not be finished. This allows validation of the cell value. The developers can always force the former behavior (no discrimination between deselections) by using the [AL\\_SetEntryOpts](#) call.

### Compatibility Note — AL ExitCell and AL Cell deselect action become AL ExitCell and AL Cell Validate

In versions of AreaList Pro previous to 7.9, there were two constants: [AL ExitCell](#) action and [AL Cell Deselect](#) action. It appears that [AL ExitCell action](#) was triggered when cell editing had to end, while [AL Cell Deselect action](#) would mean that the callback method can refuse the end of editing, but this was not implemented consistently, and AreaList Pro manual did not clearly state the difference between the events either. The manual stated that returning **False** from exit callback meant that the area refused to end editing, but AreaList Pro often ignored the return value.

Since version 7.9, AreaList Pro has changed the meaning and handling of these events as follows:

- [AL ExitCell action](#) (mode 7) means that some event occurred that requires the end of an editing session. The return values will be honored and editing will not end if callback returns **False**. For example it means that if, during editing, the user tries to scroll the area, and exit callback returns **False**, editing will not be terminated and the area will not scroll.
- [AL Cell validate action](#) (mode 8) means that 4D requested the validation of the area data, because it is going to execute some object or form method, but 4D does not want to remove focus from AreaList Pro area and editing will not end. The return value is not important, as editing will not end whatever is returned. Note that this event will occur if user clicks on non-focusable object, including buttons, popups etc. If the executed method requires that editing ends before the method is executed, it has to call [AL\\_ExitCell](#) at its beginning.



## Using the Callback Methods

For example, if a cell is currently edited and the user clicks on an enterable 4D field in the layout, the exit callback method will be called with `entryFinishedMode` parameter set to the value 7 ([AL ExitCell action](#)). If the callback method returns **False**, then the user action will be denied and the cursor will remain in the current cell of the AreaList Pro area. If the callback method returns **True**, the cursor will move to the object that was clicked.

If the user clicks on a non-focusable object such as a standard MacOS button, the exit callback method will be called with `entryFinishedMode` parameter set to the value 8 ([AL Cell validate action](#)) and the cursor will remain in the current cell of the AreaList Pro area no matter the result returned by the callback method.

---

**Warning: the row containing the currently edited data must not be deleted. No row should be deleted from any end edit type callback method.**

---

## Event Callback Interface

AreaList Pro contains an event management interface which can be used in place of the former **During (On Plug in Area)** event. When using the callback interface, 4D will no longer have the interference previously plagued by the various 4D revisions.

The callback interface is an alternate method for responding to AreaList Pro events. The [On Plug in Area](#) method and [AL\\_GetLastEvent](#) command (formerly **ALProEvt** variable) can also be used as in older versions. See [Determining the User's Action on an AreaList Pro Object](#) and [Event Callback vs Object Method](#).

## Edit Menu Callback

[AL\\_SetEditMenuCallback](#) provides an interface of overriding the default behavior when working with the 4D edit menu. You will have the option of overriding an 4D edit action for a given AreaList Pro area, providing an extensive customization interface when using Edit menu.

Some examples of how the edit menu callback interface can be used:

- You can modify the data which is placed on the clipboard to include an additional information you wish, such as header or footer data, using the [AL\\_GetHeaders](#) and [AL\\_GetFooters](#) routines, followed by calling 4D native clipboard routines.
- You can also trap information pasted to an AreaList Pro area, providing an interface which might take column row data copied from Excel and paste into AreaList Pro area, updating existing data (inserting or removing rows as necessary).

[AL\\_SetCellText](#) will set the currently highlighted cell text which can be obtained during Edit menu callback.

[AL\\_GetCellText](#) will return the currently highlighted cell text which can be obtained during Edit menu callback.

### Calculated Column Callback

A 4D callback may be attached to a specific column. When information is needed for this column, AreaList Pro will execute the callback to allow you to fill the column with data. This allows the display of data calculated from one or more fields as well as any ad hoc data that is desired.

Parameter	Description
\$1	Reference of AreaList Pro object on layout
\$2	Column number
\$3	Type of data in this column
\$4	Pointer to temporary 4D array
\$5	First record for which to calculate cell
\$6	Number of cells to calculate in column

The first three parameters are not absolutely necessary to determine how to fill the column. They are provided to give you more flexibility in the implementation of the callback method.

- The first parameter is the **areaRef**. This gives you the ability to use this callback method for more than one AreaList Pro object.
- The second parameter is the column number. This gives you the ability to use this callback method for many columns within a AreaList Pro object.
- The third parameter is the type of data in the column.

The last three parameters are absolutely necessary.

- The fourth parameter is a pointer to one of the temporary 4D arrays declared in the *Compiler\_ALP* method. This is where you will load the data to be displayed in the column.
- The fifth parameter is the number of the first cell that needs to be filled in the column. This is the same as the selected number of the row that contains this cell.
- The sixth parameter is the number of cells to be filled in the column.

---

***You must declare all six parameters (\$1 to \$6) in the calculated column callback. If any of these parameters are not declared, you will get an error when compiling the database.***

---

You must use the following declarations in your callback method:

**C\_LONGINT**(\$1;\$2;\$3;\$5;\$6)

**C\_POINTER**(\$4)

See [Setting a Calculated Column](#) for details.



# Commands

## AL\_SetMainCalls

(areaRef:L; areaEnteredMethod:S; areaExitedMethod:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ areaEnteredMethod	string	4D project method called when object is entered
→ areaExitedMethod	string	4D project method called when object is exited

**AL\_SetMainCalls** is used to set callback methods that are used when entering and exiting the AreaList Pro object.

**areaRef** — AreaList Pro area reference.

**areaEnteredMethod** — This method will be called whenever the AreaList Pro object is entered. If this is a null string then no method will be called.

The **areaEnteredMethod** project method is passed one parameter. This parameter is a long integer that corresponds to the AreaList Pro object on the layout.

---

*If the AreaList Pro object is the first object in the entry order, when the layout containing the AreaList Pro object is first opened, the areaEnteredMethod will not be called. This is because 4D gives the event to AreaList Pro (to inform it that it is to be the active object when the layout is opened) prior to the execution of the layout's method. If you want to take action based upon this active object, then call the areaEnteredMethod from the On load phase in your 4D code.*

---

**areaExitedMethod** — This method will be called whenever the AreaList Pro object is exited. If this is the null string then no method will be called.

The **areaExitedMethod** project method is passed one parameter. This parameter is a long integer that corresponds to the AreaList Pro object on the layout.

Some of the uses of these callbacks are as follows:

- Enabling buttons or other variables that pertain to the AreaList Pro object from the **areaEnteredMethod**. With 4D 2003 and earlier versions, you had to use interprocess buttons or variables and call **REDRAW WINDOW** or **CALL PROCESS (-1)** to update them. This is not needed with 4D 2004 and above.
- Disabling buttons or other variables that pertain to the AreaList Pro object from the **areaExitedMethod**. With 4D 2003 and earlier versions, you had to use interprocess buttons or variables and call **REDRAW WINDOW** or **CALL PROCESS (-1)** to update them. This is not needed with 4D 2004 and above.
- Call [AL\\_GotoCell](#) from the **areaEnteredMethod** to initiate data entry when the object is entered.

## Using the Callback Methods

Example:

`Set up area entered and area exited callbacks

**AL\_SetMainCalls**(eList;"AreaEnteredMethod";"AreaExitedMethod")

`AreaEnteredMethod, area entered callback method

**C\_LONGINT**(\$1)

**AL\_GotoCell**(\$1;1;1) `initiate data entry on the first cell in the first column

**ENABLE BUTTON**(bChangeSub)

**ENABLE BUTTON**(bAltRowBkd)

**CALL PROCESS**(-1)

`AreaExitedMethod, area exited callback method

**C\_LONGINT**(\$1)

**DISABLE BUTTON**(bChangeSub)

**DISABLE BUTTON**(bAltRowBkd)

**CALL PROCESS**(-1)

---

## AL\_SetCallbacks

(areaRef:L; entryStartedMethod:S; entryFinishedMethod:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ entryStartedMethod	string	4D method called when entry in cell started
→ entryFinishedMethod	string	4D method called when entry in cell finished

**AL\_SetCallbacks** is used to set callback methods that are used with data entry. Please read the section [Using Callback Methods During Data Entry](#) for more information.

**entryStartedMethod** — This project method will be called whenever data entry is started in a cell or when a popup menu is clicked. If this is the null (empty) string then no method will be called.

The **entryStartedMethod** is passed two parameters. The first parameter is a longint that corresponds to the AreaList Pro object on the layout. The second parameter is a longint that reports what action (mode) caused data entry to be started in the cell.

## Using the Callback Methods

For a list of the possible values of the second parameter, see the table below:

Constant	Value	Entry Mode
AL Click action	1	Click in Cell
AL Tab key action	2	Tab
AL Shift_Tab key action	3	Shift-Tab
AL Return key action	4	Return
AL Shift_Return key action	5	Shift-Return
AL GotoCell action	6	<a href="#">AL_GotoCell</a>
	7	Not used
	8	Not used
AL SkipCell action	9	<a href="#">AL_SkipCell</a>
AL Other cell popup action	10	Click on cell popup when cursor not already in cell
AL Active cell popup action	11	Click on cell popup when cursor already in cell

**entryFinishedMethod** — This project method will be called whenever data entry is finished in a cell or when a popup menu is released in a cell for which typed data entry has not been established. This method must be a function. It must return **True** for the value entered into the cell to be accepted and **False** for the value to be rejected. If this is the null (empty) string then no method will be called.

The **entryFinishedMethod** is passed two parameters. The first parameter is a longint that corresponds to the AreaList Pro object on the layout. The second parameter is a longint that reports what action (mode) caused data entry to be finished in the cell.

For a list of the possible values of the second parameter, see the table below.

Constant	Value	Exit Mode
AL Click action	1	Click outside cell on object
AL Tab key action	2	Tab
AL Shift_Tab key action	3	Shift-Tab
AL Return key action	4	Return
AL Shift_Return key action	5	Shift-Return
AL GotoCell action	6	<a href="#">AL_GotoCell</a>
AL ExitCell action	7	<a href="#">AL_ExitCell</a> or “hard deselect”
AL Cell validate action	8	Deselect the cell (“soft deselect”)
	9	Not used
AL Other cell popup action	10	Cell popup released when cursor not already in cell
AL Active cell popup action	11	Cell popup released when cursor already in cell

When a cell is entered the data will be copied into the zero element of the array being displayed in the column. When the **entryFinishedMethod** is executed the array element corresponding to the cell will already be updated with the new value that was entered.

## Using the Callback Methods

Among the possible situations and responses that may occur are the following:

- The data is valid. Set `$0:=True` to complete data entry for the cell.
- The data is invalid. Copy the old data from the zero element to the array element corresponding to the cell. Set `$0:=True` to complete data entry for the cell.

For example:

```
aFname{vRow}:=aFname{0} `reset the cell contents to their original state
$0:=True
```

- The data is invalid. Inform the user that the data is invalid. Set `$0:=False` to force the user to remain in the cell and enter another value.
- The data is invalid. Inform the user that the data is invalid. Modify the cell contents, call [AL\\_GotoCell](#) to go to the current cell, and set `$0:=True`. This achieves the same effect as rejecting the entry, but allows the cell contents to be modified.

Examples:

```
aFname{vRow}:=aFname{0} `reset the cell contents to their original state
AL_GotoCell(eList;vColumn;vRow) `go to the same cell
$0:=True
```

```
`Don't install an entry started method, do install an entry finished method
AL_SetCallbacks(eList;"";"EntryDoneMethod")
```

See also [Compatibility Note — AL ExitCell and AL Cell deselect action become AL ExitCell and AL Cell Validate](#).

---

## AL\_SetEventCallback

(areaRef:L; callbackMethod:S; flag:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ callbackMethod	string	A valid 4 <sup>th</sup> Dimension method which will be called during AreaList Pro event execution
→ flag	integer	Handling flag

**AL\_SetEventCallback** provides an alternate method for dealing with all the events which can be triggered when working with AreaList Pro areas. The historical event triggering system ([AL\\_GetLastEvent](#) command, formerly **ALProEvt** variable) still works, but the callback method provides a more generic developer control for precise event handling.

---

**Note that some 4D commands can only be called with the [On Plug in Area](#) method and [AL\\_GetLastEvent](#) command (formerly **ALProEvt** variable). See [User Action Commands](#).**

---

## Using the Callback Methods

**callbackMethod** — A valid 4<sup>th</sup> Dimension method which will be called during AreaList Pro event execution.

**flag** — Compatibility flag that defines how and when the area object method is executed:

- 0** — compatible mode, the area object method and the form method are executed in AreaList Pro 7.9 and above the same way as in earlier versions
- 1** — the area object method and the form method are executed except for events that require posting cmd-\ (single-click and scroll events)
- 2** — the area object method and the form method are not executed at all (use the event callback)

---

***Control-click events are reported to the event method immediately (when the mouse is still down).***

---

The following example installs an event callback method which is executed during any AreaList Pro event:

```
$err:=AL_SetEventCallback(areaRef;"CallbackMethod";2)
```

The **callbackMethod** is a 4D project method with the following declarations:

```
C_LONGINT($0) `object method and form method will not be executed if 0  
C_LONGINT($1;$area) `AreaList Pro area  
C_LONGINT($2;$alpEvent) `AreaList Pro event  
C_LONGINT($3;$alpEventMod) `event modifier — unused now, may be used later for passing  
    additional info about the event  
C_LONGINT($4;$col) `column — last clicked column  
C_LONGINT($5;$row) `row — last clicked row  
C_LONGINT($6;$modifiers) `modifiers  
C_STRING(255;$7;$tip) `tip string  
C_STRING(255;$8;$areaName) `plug-in area name (see AL_SetAreaName)
```

The \$0 result code can be set to 0 or 1:

- 0** — indicates that the event was handled by the callback method
- 1** — indicates that the event was not handled by the callback method, the object method or form method will be executed with the \$2 event as [AL\\_GetLastEvent](#) (formerly **ALProEvt variable**).

---

***\$2 contains the same AreaList Pro event as passed by the **AL\_GetLastEvent** command (formerly **ALProEvt variable**) to the area object method or form method.***

---

The handling of events is similar in both cases, so if you want to have both single and double-clicks reported, AreaList Pro will still wait for the double-click time to decide if it received a single or a double-click.

See [User Action Commands](#) for additional information about event codes.

### AL\_SetEditMenuCallback

(areaRef:L; callbackMethod:S) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ callbackMethod	string	4D project method called when the Edit menu is used
← resultCode	longint	Result code

**AL\_SetEditMenuCallback** provides an interface of overriding the default behavior when working with the 4D edit menu.

**callbackMethod** — Desired Edit menu callback method to use for the given area.

The edit menu callback method will receive the area reference (parameter \$1) and the edit event (parameter \$2).

Edit events are received in the longint \$2 parameter, using bitwise operators to extract subevents.

When you wish to act on several edit menu items, the value is the result of a bitwise operation.

**For more information on working with bitwise values, please refer to the 4<sup>th</sup> Dimension Command Reference**

Edit menu constant	Value	Description	Bit position constant	Value
AL Edit Menu Setup Mask	65536	AreaList Pro is setting up edit menu, before menu is displayed	AL Edit Menu Setup Bit	16
AL Edit Menu Entry Mask	32768	AreaList Pro cell editing is in progress	AL Edit menu Entry Bit	15
AL Edit Menu All Items Mask	127	AreaList Pro is using all possible edit menu items		
AL Edit Menu Select All Mask	64	Reference to Edit menu select all menu item	AL Edit Menu Select All Bit	6
AL Edit Menu Clear Mask	32	Reference to Edit menu clear menu item	AL Edit Menu Clear Bit	5
AL Edit Menu Paste Mask	16	Reference to Edit menu paste menu item	AL Edit Menu Paste Bit	4
AL Edit Menu Copy Mask	8	Reference to Edit menu copy menu item	AL Edit Menu Copy Bit	3
AL Edit Menu Cut Mask	4	Reference to Edit menu cut menu item	AL Edit Menu Cut Bit	2
AL Edit Menu Redo Mask	2	Reference to Edit menu redo menu item	AL Edit Menu Redo Bit	1
AL Edit Menu Undo Mask	1	Reference to Edit menu undo menu item	AL Edit Menu Undo Bit	0

## Using the Callback Methods

The Edit menu callback returns a longint \$0 result, containing details of what action(s) should take place

When working with the callback, you can return a series of values. If you want AreaList Pro to handle the Edit menu selection, return a value of zero (0). If wish to customize the result, return AL Edit Menu Handled Mask (tells AreaList Pro that we handled the menu in the callback method).

Edit menu constant	Value	Description	Bit position constant	Value
AL Edit Menu Handled Mask	131072	Tells AreaList Pro that the callback method handled the Edit menu operation	AL Edit Menu Handled Bit	17

For example, If you wish to tell AreaList Pro to enable only the Copy and Select All menu items (overriding default settings), you would return the following result.

```
$AL_Result:=( $AL_Result & AL Edit Menu All Items Mask ) ?+ AL Edit Menu Handled Bit
$AL_Result:=$AL_Result ?+ AL Edit Menu Copy Bit `enable copy menu item
$AL_Result:=$AL_Result ?+ AL Edit Menu Select All Bit `enable select all menu item
$0:=$AL_Result
```

### Edit Menu Callback Framework

The callback method should include the following framework parameter declaration. Each parameter must be declared in every callback method and a result value must be returned. Failure to properly declare variables or return the result variable will produce a compiler error when using database in compiled mode.

```
`$0: result
`0 — AreaList Pro will handle event(default)
`>0 — callback handled event
`$1: AreaList Pro area reference
`$2: AreaList Pro Edit event
`$3: Undo (unused, but required internally)
C_LONGINT ($0;$AL_Result)
C_LONGINT ($1;$AL_Area)
C_LONGINT ($2;$AL_Event)
C_TEXT ($3;$AL_Undo)
$AL_Result:=0 `default result, AreaList Pro will handle event
$AL_Area:=$1
$AL_Event:=$2
$AL_Undo:=$3
Case of
...
End case
$0:=$AL_Result
```

### AL\_SetCalcCall

(areaRef:L; columnNumber:I; calcCallback:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column number
→ calcCallback	string	4D method called ot fill row(s) of a calculated column

**AL\_SetCalcCall** is used to set a callback method for a calculated column.

**columnNumber** — This parameter specifies the column on which to attach the **calcCallback** method.

**calcCallback** — This method will be called whenever row(s) need to be filled in a calculated column. If this is an empty string then no method will be called.

The first two parameters (\$1 and \$2) passed to this callback method are **areaRef** and **columnNumber**. Therefore, if desired, the same callback can be used for more than one AreaList Pro object and for many columns in an object.

For information on how to write a calculated column callback, see the section [Calculated Column Callback](#).

Example:

```
`Set calculated callback method for column 3  
AL_SetCalcCall(eList;3;"CalcColCallback")
```



# Field and Record Commands

AreaList Pro uses the **SELECTION RANGE TO ARRAY** command in 4D to get the records for display.

Up to 512 fields (columns) can be displayed in an AreaList Pro object.

You can use the Advanced Properties Dialog to configure the fields to display in an AreaList Pro object. Please read the section [Configuring AreaList Pro Using the Advanced Properties Dialog](#) for more information.

## Using the Field Display Capability

### Temporary Arrays

AreaList Pro internally uses interprocess 4D arrays to get the record data from 4<sup>th</sup> Dimension. These arrays must be declared in 4D. A text file has been included that contains these declarations. Simply create a 4D global method named *Compiler\_ALP* and copy these declarations into it. There is no need to call this method from your 4D code, AreaList Pro will call it for you. This method must exist whether your database is interpreted or compiled.

Do not access the data within these temporary arrays. These arrays are for AreaList Pro's internal use only and their contents may change at any time.

Only 30 arrays of each of the 9 data types that AreaList Pro supports are declared. If you will be displaying more than 30 fields of a certain type, then you must add more declarations within the *Compiler\_ALP* project method.

Conversely, you may remove some of these declarations if you never display fields (or display very few fields) of a certain type. Be very careful (when adding or removing declarations) to follow exactly the syntax of the existing declarations.

### Arrays and Fields

To change the display from arrays to fields, first call [AL\\_RemoveArrays](#) to remove all of the arrays before calling any field commands.

To change the display from fields to arrays, first call [AL\\_RemoveFields](#) to remove all of the fields before calling any array commands.

---

***Arrays and fields may not be displayed together in the same AreaList Pro object. If arrays are displayed in an object, then the field commands will be ignored. Conversely, if fields are displayed in an object, then the array commands will be ignored.***

---

# Compatibility Note — Field Display and Callbacks

With AreaList Pro 7.9 and above, if you are using AreaList Pro to display fields and are performing custom actions via AreaList Pro's exit callback (see [AL\\_SetCallbacks](#)) you should no longer be executing code which will change the current selection as AreaList Pro now maintains the current record status when the exit callback method is invoked.

For example, with previous versions you had to do something like the following in your exit callback code:

```
AL_GetCurrCell($1;$row;$col)
GOTO SELECTED RECORD ([[Customers];$row)
```

This is no longer required and if the code is still making this type of call that will change the current selection, you will lose your edits.

---

*When inside the callback method, you can obtain the previous table value by using the 4D Command Old.*

---

```
$prevValue:=Old ([[Customers]Zip)
If ($prevValue# [[Customers]Zip)
    $ret:=ZC_LookupZip ([[Customers]Zip;-> [[Customers]City; [[Customers]State)
End if
```

## Setting a Calculated Column

The commands [AL\\_SetFields](#) and [AL\\_InsertFields](#) are used both to set fields to be displayed and to set up calculated columns.

If the **fieldNum** parameter contains an integer greater than or equal to 1, the column will display the field represented by that number.

If the **fieldNum** parameter contains an integer less than or equal to 0, the column will display calculated data. The absolute value of **fieldNum** will determine the type of data to be displayed in the column. The following table shows the data types that may be displayed in a calculated column.

Constant	Value
<u>Is Alpha Field</u>	0
<u>Is Real</u>	1
<u>Is Text</u>	2
<u>Is Picture</u>	3
<u>Is Date</u>	4
<u>Is Boolean</u>	6
<u>Is Integer</u>	8
<u>Is LongInt</u>	9
<u>Is Time</u>	11

For example, to display a calculated column of type Real, pass Is Real (-1) in the **fieldNum** parameter.

### Setting the Callback Method

Use the [AL SetCalcCall](#) command to set the callback method for a column.

---

***AreaList Pro will dimension the temporary array before invoking the calculated column callback. There is no need to do it in the callback itself.***

---

The following is an example of a calculated callback method. It merely calculates an employee's one year anniversary by adding 365 to their hire date (this obviously does not take into account leap years, but is sufficient as an example).

```
`CalcColCallback
`$1: Area reference (AreaList Pro longint reference)
`$2: Column number
`$3: Type of data in this column
`$4: Pointer to temporary 4D array
`$5: First record for which to calculate cell
`$6: Number of cells to calculate in column
`Declare the parameters
C_LONGINT($1;$2;$3;$5;$6) `these must be declared
C_POINTER($4) `this must be declared
C_LONGINT($i)
ARRAY DATE($aHireDate;0)
SELECTION RANGE TO ARRAY($5;$5+$6-1;[Employee]Hire Date;$aHireDate)
For($i;1;$6)
    $4->{$i}:=$aHireDate{$i}+365
End for
```

### Sorting

Calculated columns will not be sorted when their column header is clicked upon. However, if the `userSort` option of [AL SetSortOpts](#) command is set to 2, "Bypass the user sort buttons", and the column header of a calculated column is clicked upon, the AreaList Pro event callback (or area/form method) will run, with a \$2 event code of -1 returned to the callback method (or [AL GetLastEvent](#) command, formerly `ALProEvt` variable). See [Determining the User's Action on an AreaList Pro Object](#).

The command [AL SetSort](#) command will not allow sorting of calculated columns.

### Enterability

Calculated columns are not enterable by any method, including using the [AL GotoCell](#) command.

### Time Data

Time data will be converted to a longint since this is how it is stored internally by 4D.

# Displaying 4D Fields

## Fields from Related One Tables

Fields from a main table and from related one tables may be displayed in the same AreaList Pro object. See the commands [AL\\_SetFile](#) and [AL\\_SetFields](#) for further information about displaying fields from related one tables.

## Redraw and Scrolling

When 4D fields are displayed, the visible rows are cached (held in memory). This is done to improve redraw speed. Every field within the visible rows are held in memory so horizontal scrolling is as fast as when displaying arrays. Vertical scrolling will be slower since the records not in view have to be retrieved from 4D.

## Type-ahead

Keyboard type-ahead will be disabled when displaying fields.

## Copy Rows to the Clipboard

Copying rows to the clipboard will not be allowed when displaying fields. The “Copy” menu item will be disabled when fields are displayed.

## Enterability

Columns containing fields from a related one table will not be enterable either by typing or by using popups.

## Dragging

When displaying arrays, AreaList Pro will rearrange the rows automatically when the user drags a row within the list. When displaying fields, AreaList Pro will not rearrange the rows automatically when the user drags a row within the list. Thus the `moveWithData` option of [AL\\_SetRowOpts](#) and the `moveWithData` of [AL\\_SetCellOpts](#) do not apply when fields are displayed and the user drags a row within the list.

## Sorting

- indexed fields will be bold in the Sort Editor
- fields from related one tables will be dimmed in the Sort Editor
- columns containing fields from a related one table will not be sorted when their column header is clicked upon

When fields are displayed the `moveWithData` of [AL\\_SetRowOpts](#) will be ignored when sorting. The row style and color information will not move with the row when the AreaList Pro object is sorted.

When fields are displayed the `moveWithData` of [AL\\_SetCellOpts](#) will be ignored when sorting.

## Field and Record Commands

The cell style and color information will not move with the cell when the AreaList Pro object is sorted.

When the `userSort` option of [AL\\_SetSortOpts](#) is set to 3 and fields are being displayed, only columns containing indexed fields may be sorted by clicking on their column header.

---

*AreaList Pro uses 4<sup>th</sup> Dimension's sorting routines when sorting fields. 4D only uses indexes when performing a single level sort. Indexes are ignored when performing a multiple level sort. Therefore, when fields are being displayed, it would be a good idea to restrict access to the AreaList Pro sort editor when the selection contains several thousand records.*

---

## Maximum Number of Records Displayed

AreaList Pro supports a maximum of 2 billion (exactly 2 147 483 647) records displayed in an AreaList Pro object.

You can also display a selection with any desired number of records up to this limit, using [AL\\_SetSubSelect](#) to specify what record range within the current selection you wish to display.

## Performance Issues When Displaying Fields

When AreaList Pro displays fields, the automatic column sizing algorithm uses only the first 20 records (or less, if the selection contains less than 20 records) in the selection. These records are always read regardless of whether the columns are automatically or manually sized.

Therefore there is no performance penalty using the automatic column sizing algorithm when displaying fields. See [Performance Issues with Formatting Commands](#) for more information.

## Commands

---

### AL\_SetFile

(areaRef:L; tableNum:I) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ tableNum	integer	Number of 4D table
← resultCode	longint	Result code

**AL\_SetFile** tells AreaList Pro what table is the main table from which to display records.

This command is only necessary if the field to be displayed in column one is not from the main table, but from a related one table.

**AL\_SetFile** must be called before any fields have been set, otherwise it will be ignored. If this command is not called, then AreaList Pro will use the table of the field displayed in column one as the main table.

## Field and Record Commands

**resultCode** — The possible values are:

Value	Result Code	Action
0	No error	
1	Not a table	Check to make sure that the table represented by <b>tableNum</b> does exist

Example:

```
$result:=AL_SetFile(eList;Table (->[People]))
```

---

## AL\_SetFields

(areaRef:L; tableNum:I; columnNumber:I; numFields:I; fieldNum1; ...; fieldNumN:I)  
→ resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ tableNum	integer	Number of 4D table
→ columnNumber	integer	Column at which to set the first field
→ numFields	integer	Number of fields to set (up to 15)
→ fieldNum	integer	Number of 4D field
← resultCode	longint	Result code

**AL\_SetFields** tells AreaList Pro what fields to display. Up to fifteen fields can be set at a time. Any 4D field type can be used except sub-tables.

Fields from related one tables may also be displayed (see [AL\\_SetFile](#)). A separate call to **AL\_SetFields** must be made to set these fields. To display a related one field, pass the table number of the related one table in the **tableNum** parameter.

**resultCode** — The possible values are:

Constant	Value	Action
AL No error in fields	0	
AL Not a file error	1	Check to make sure that the table represented by <b>tableNum</b> does exist
AL Not a field error	2	Check to make sure that the field represented by <b>fieldNum</b> does exist
AL Wrong type field error	3	Sub-tables are not allowed
AL Max fields exceeded error	4	512 fields is the maximum
AL Wrong 4D vers for fields	5	(obsolete)
AL Low memory field error	6	Increase 4D's RAM partition

## Field and Record Commands

Examples:

`Set up the eList AreaList Pro object with 5 fields, all from the same table

```
$error:=AL_SetFields(eList;Table(->[People]);1;5;Field(->[People]First Name);  
Field(->[People]Last Name);Field(->[People]Salary);Field(->[People]Arrival);Field(->[People]Male))
```

`Set up the eList AreaList Pro object with 4 fields, the third one from a related table

```
$error:=AL_SetFields(eList;Table(->[People]);1;2;Field(->[People]First Name);  
Field(->[People]Last Name))
```

```
$error:=AL_SetFields(eList;Table(->[Companies]);3;1;Field(->[Companies]Company Name))
```

```
$error: AL_SetFields(eList;Table(->[People]);4;1;Field(->[People]Salary))
```

`Set up the eList AreaList Pro object with 4 fields, the first one from a related table

```
$error:=AL_SetFile(eList;Table(->[People])) `set the main table since the field to be set in column one is  
not from the main table, but from a related one table
```

```
$error:=AL_SetFields(eList;Table(->[Companies]);1;1;Field(->[Companies]Company Name))
```

```
$error:=AL_SetFields(eList;Table(->[People]);2;3;Field(->[People]First Name);  
Field(->[People]Last Name); Field(->[People]Salary))
```

---

## AL\_GetMode

(areaRef:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← resultCode	longint	Result code

**AL\_GetMode** will return the type of display method you are using for the supplied AreaList Pro areaRef area.

**result** — Returns one of the following results:

**0** — a value of zero will be returned if you are displaying arrays

**1** — a value of one will be returned if you are displaying fields

The following will build an AreaList Pro area based on field references from a parent and related table.

```
$ret:= AL_SetFields(eList;Table(->[People]);1;2;Field(->[People]FirstName);Field(->[People]LastName))
```

```
$ret:= AL_SetFields(eList;Table(->[Companies]);3;1;Field(->[Companies]Company Name))
```

```
$ret:= AL_SetFields(eList;Table(->[People]);4;1;Field(->[People]Salary))
```

Then, we'll use the **AL\_GetMode** routine to determine the type of objects we are using to build the list.

```
$ret:=AL_GetMode(eList) `a value of 1 will be returned as we are using fields
```

### AL\_GetTable

(areaRef:L) → tableNumber:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← tableNumber	longint	Primary table number

**AL\_GetTable** will return the primary table number you are using for the supplied AreaList Pro reference. This is the table number supplied by [AL\\_SetFields](#), [AL\\_InsertFields](#) or [AL\\_SetFile](#).

**tableNumber** — Returns the primary table number.

The following will build an AreaList Pro area based on field references from a parent and related table:

```
$ret:= AL_SetFields(eList;Table(->[People]);1;2;Field(->[People]FirstName);Field(->[People]LastName))  
$ret:= AL_SetFields(eList;Table(->[Companies]);3;1;Field(->[Companies]Company Name))  
$ret:= AL_SetFields(eList;Table(->[People]);4;1;Field(->[People]Salary))
```

Then, we'll use the **AL\_GetTable** routine to determine the primary table.

```
$ret:=AL_GetTable(eList) `the table number associated to [People] table will be returned.
```

### AL\_GetFields

(areaRef:L; tableArray:X; fieldArray:X) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← tableArray	longint array	List of table numbers
← fieldArray	longint array	List of field numbers
← resultCode	longint	Result code

**AL\_GetFields** will return an array of table and field numbers used to build the list reference based on 4D field references (see [AL\\_SetFields](#)).

**tableArray** — A valid 4<sup>th</sup> Dimension longint array which will contain the list of table numbers.

**fieldArray** — A valid 4<sup>th</sup> Dimension longint array which will contain the list of field numbers.

**result** — Returns one of the following results:

- 50 — Parameter error (array of wrong type, must be longint arrays)
- 1 — Wrong mode (see [AL\\_GetMode](#))



## Field and Record Commands

The following example will build an AreaList Pro area based on field references from a parent and related table:

```
$ret:= AL_SetFields(eList;Table(->[People]);1;2;Field(->[People]FirstName);Field(->[People]LastName))  
$ret:= AL_SetFields(eList;Table(->[Companies]);3;1;Field(->[Companies]Company Name))  
$ret:= AL_SetFields(eList;Table(->[People]);4;1;Field(->[People]Salary))
```

Then, we'll use the **AL\_GetFields** routine to return arrays of table and field numbers.

```
ARRAY LONGINT(aiAL_TableNo;0)  
ARRAY LONGINT(aiAL_FieldNo;0)  
$ret:=AL_GetFields(eList;aiAL_TableNo;aiAL_FieldNo)
```

aiAL\_TableNo and aiAL\_FieldNo will contain 3 entries each.

---

## AL\_InsertFields

(areaRef:L; tableNum:I; columnNumber:I; numFields:I; fieldNum1:I ... fieldNumN:I)  
→ resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ tableNum	integer	Number of 4D table
→ columnNumber	integer	Column at which to set the first field
→ numFields	integer	Number of fields to set (up to 15)
→ fieldNum1; ...;fieldNumN	integer	Number of 4D field(s)
← resultCode	longint	Result code

**AL\_InsertFields** functions the same as [AL\\_SetFields](#), except that the fields are inserted before columnNumber.

All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding field.

Example:

```
`Add a column to display the first name  
$result:=AL_InsertFields(eList;Table(->[People]);4;1;Field(->[People]First Name))
```

### AL\_RemoveFields

(areaRef:L; columnNumber:I; numFields:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column at which to remove the first field
→ numFields	integer	Number of fields to remove (up to 512)

**AL\_RemoveFields** is used to remove fields from AreaList Pro. **numFields**, beginning at **columnNumber**, will be removed from the list. All subsequent columns will maintain their settings. In other words, any header text, column styles, etc. will stay with their corresponding field.

Example:

`Remove two columns, beginning at column #4

**AL\_RemoveFields**(eList;4;2)

### AL\_UpdateFields

(areaRef:L; updateMethod:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ updateMethod	integer	Method to use to update the AreaList Pro object

**AL\_UpdateFields** is used to update AreaList Pro. Use this command whenever any records of the fields being displayed are changed (records added, deleted, or modified), but the fields themselves remain the same. **AL\_UpdateFields** must be called after modifying the fields and before any other setup commands (sorting, formatting, etc.).

**updateMethod** — This parameter tells AreaList Pro how to update the AreaList Pro object **areaRef**. The possible values are:

Constant	Value	Description	When to Use
AL Refresh fields	0	Refresh the AreaList Pro object, but don't update any records, and don't recalculate any values	When changes are made to formatting, color, styles, etc.
AL Refresh and update fields	1	Refresh the AreaList Pro object, and update the visible records, but don't recalculate any values	When changes are made to the contents of the records shown in the visible rows
AL Recalculate fields	2	Rescan all visible rows and recalculate all applicable heights, widths, and other related values. The scroll position, and row or cell selection will be reset	If column or row resizing is necessary, or you have added or deleted records pertaining to the displayed fields. Also if you show or hide either scrollbar, the headers, or footers

### AL\_SetSubSelect

(areaRef:L; firstRecord:L; numRecords:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ firstRecord	longint	First record to display
→ numRecords	longint	Number of records to display

**AL\_SetSubSelect** is used to tell AreaList Pro to display a different subselection of records from the current selection.

This command will have the same effect on the AreaList Pro object as calling [AL\\_UpdateFields](#) with **updateMethod** set to 2, in addition to changing the subselection of records to be displayed. Thus if this command is called, there is no need to also call **AL\_UpdateFields**.

**firstRecord** — This parameter is used to set the first record in the selection to be displayed in the AreaList Pro object. If **firstRecord** is greater than or equal to the number of records in the selection, then it will be set to the last record in the selection. The default is 1.

**numRecords** — This parameter is used to set the number of records in the selection to be displayed in the AreaList Pro object. The possible values are:

Value	Description
>= 0	Display this number of records
-1	Display the number of records from <b>firstRecord</b> until the end of the selection

If **numRecords** is greater than the number of records from **firstRecord** until the end of the selection, then **numRecords** will be set to the number of records from **firstRecord** until the end of the selection.

If this command is not called, then **firstRecord** will be set to 1 and **numRecords** will be set to the number of records in the selection.

Example:

`Set up the eList AreaList Pro object to display 10,000 records beginning at record 5001

**AL\_SetSubSelect**(eList;5001;10000)

# Enterability

## Initiating Data Entry

The method for initiating entry to a cell, and for selecting rows, is set with the `entryMode` parameter of `AL_SetEntryOpts`.

Initiating entry can be done in any one of eight different ways, each of which also determines the method for selecting rows. See [AL\\_SetEntryOpts](#) for complete information.

Enterability for a column is set using [AL\\_SetEnterable](#).

## Entering Data

The capability to edit data during typed data entry is initiated automatically, and no programming is necessary to invoke these functions.

When data entry is initiated on an AreaList Pro cell, the array contents for the element corresponding to that cell are copied to the zero element of the same array.

Since this element is usually never used, it makes a convenient storage place for the data in case you wish to revert to the old value; however, you should take care not to use this zero array element elsewhere in your code while data entry is in progress.

---

***When fields are displayed you are responsible for saving the contents of the field.***

---

Two commands, [AL\\_SetCellHigh](#) and [AL\\_GetCellHigh](#), can be used to set the highlighted range of characters in the data entry cell, or get the range of characters highlighted by the user, respectively. `AL_SetCellHigh` can also be used to set the insertion point between two characters.

After the user ends data entry on a particular array element, [AL\\_GetCellMod](#) can be used to determine if the data has been altered. `AL_GetCellMod` and `AL_GetCellHigh` can only be used within an entry finished callback method. See [Using Callback Methods During Data Entry](#).

Other programmable data entry specifications include the use of the Return key for movement during data entry, or insertion of a carriage return character into the text data being entered. This is controlled using the `allowReturn` parameter of [AL\\_SetEntryOpts](#). Please read the section [Moving the Current Entry Cell](#) for more information.

You can also specify that seconds be displayed (hh:mm:ss) when the user is entering time data through the use of the `displaySeconds` parameter of `AL_SetEntryOpts`.

For boolean data type arrays, two data entry methods can be specified: a checkbox or radio buttons. [AL\\_SetEntryCtrls](#) is used to specify which of these controls is used, and to which column it applies.

## Filters

In order to use data entry filters [AL\\_SetFilter](#) must be used on a per column basis.

Standard 4D filter strings can be used, except that placeholders are not supported and will be ignored. Pre-defined styles may not be used for data entry filters.

## Click and Hold Data Entry Initiation

Cell entry will be initiated when the user clicks and holds down the mouse button for the developer determined period of time.

When you have configured AreaList Pro to allow data entry (using [AL\\_SetEntryOpts](#)) with an modifier-click (control, command, etc.) data entry will be automatically initiated when the user holds down the mouse button.

This interface will allow you to create an interface whereby users can single-click or double-click on cells and initiate data entry without requiring the defined keyboard modifier.

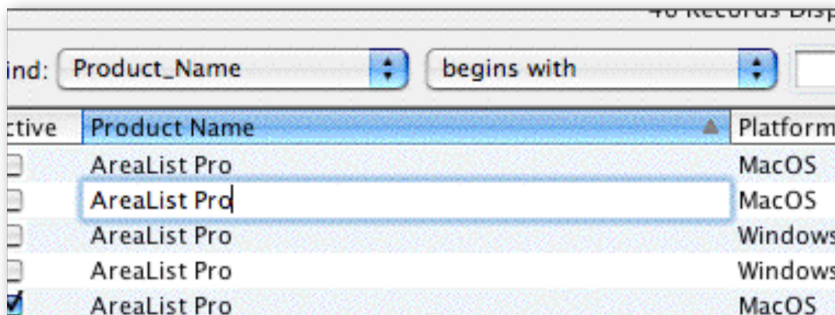
The following example has data entry configured as control-double-click, however, it will also be activated when the user has held down the mouse button in an enterable cell for one second (60 ticks) as configured in [AL\\_SetInterface](#):

**AL\_SetInterface**(eList;-1;-1;-1;-1;60) `initiate data entry after one second of holding

**AL\_SetEntryOpts**(eList;7;0)

## Entry Cell Border

AreaList Pro uses the native cell border when performing data entry. Native system commands are used on both platforms to draw a focus rectangle around the edited text.



AREALIST PRO CELL BORDER

### Popups

As an alternative to typed data entry, you can specify that a column use popup menus by using the `popupArray` parameter of [AL\\_SetEnterable](#). In this parameter, an array is passed to AreaList Pro, with which AreaList Pro will build a popup menu.

No array needs be passed to AreaList Pro for a time or date column which uses a popup menu. AreaList Pro provides specialized menus for these data types. The presence of a popup menu in a cell does not prohibit the user from entering typed data; the `enterability` parameter of [AL\\_SetEnterable](#) allows you to control whether either one or both of these data entry methods are allowed.

---

*The popup menu array must be of the same data type as the data in the column. It is important that the array used for a popup not be disposed of until it is no longer needed.*

[AL\\_SetEnterable](#) must be called when any changes are made to a `popupArray`.

---

You can optionally disable meta characters in AreaList Pro enterable popup controls, enabling you to use special characters such as "/" or "(" in menu items.

There are two different methods for disabling meta characters. The first method will be using a new parameter in [AL\\_SetInterface](#) routine, while the second method will be using new options in the [AL\\_SetEnterable](#) routine.

### Moving the Current Entry Cell

The action of the Carriage Return key is determined by the programmer using the `allowReturn` parameter of [AL\\_SetEntryOpts](#), depending upon the data entry requirements of the database.

The user's ability to control movement while in data entry can also be established with the use of the `moveWithArrows` and `mapEnterKey` parameters of this command. The `moveWithArrows` parameter will allow the user to move from cell to cell while in data entry using the four Arrow keys. `mapEnterKey` enables you to cause the Enter key to act the same way as either the Tab key or the Return key. When using this parameter, it should be noted that the Enter key is often used in 4<sup>th</sup> Dimension for other functions which may conflict with its AreaList Pro meaning.

A variety of AreaList Pro commands enable you to monitor and control movement during data entry. The current and previous data entry cells can be determined by using [AL\\_GetCurrCell](#) and [AL\\_GetPrevCell](#), respectively. Movement from cell to cell, while staying in data entry mode, can be accomplished using [AL\\_GotoCell](#). [AL\\_SkipCell](#) can be used in the entry started callback method to cause data entry on a particular cell to be skipped. Data entry can be terminated via [AL\\_ExitCell](#).

## Compatibility Note

### Adding or Deleting Rows from a Form Button

When using AreaList Pro as an included layout, it is common to have buttons in the input form to add or delete rows from the AreaList Pro area. The code that performs this type of action, if written before version 7.9, requires a small change so that the records are correctly added/deleted when using enterability. There are three different methods you can use to handle this issue.

- 1 — option one will be to modify the call to [AL\\_SetEntryOpts](#) to instruct 4D to commit the current cell when the focus is lost:

**AL\_SetEntryOpts**(eList;n+8;...) `where n is the desired data entry trigger

- 2 — option two will be to call [AL\\_ExitCell](#) in the object method of the add or delete button on each method which performs a similar action:

**AL\_ExitCell**(eCustomers)

- 3 — place a call in each of the exit callback methods to instruct AreaList Pro to exit the cell:

**AL\_ExitCell**(\$1)

## Redrawing the Display from the Callback Method

You may want to display a variable which has been updated in one of the available callback methods on the same layout as the AreaList Pro object. The variable's value will be successfully updated in the callback method, but it will not be displayed on the layout immediately. This is because 4D will not refresh the screen when a displayed value changes while a plug-in is controlling execution.

If the command **REDRAW WINDOW** is called or **CALL PROCESS** is used with the **process id** parameter set to -1, 4D will refresh all windows displaying an interprocess variable. This method requires that if a variable is updated from the callback method, then it must be an interprocess variable. In addition, the **REDRAW WINDOW** or **CALL PROCESS** commands should be executed from the callback method.

Example:

```
C_LONGINT($1;$2;$AL_Object;$Action;$i)
C_REAL($Total)
$AL_Object:=$1
$Action:=$2
$Total:=0
For($i;1;Size of array(aAmounts))
    $Total:=$Total+aAmounts{$i}
End for
◇Total:=$Total
CALL PROCESS (-1)
$0:=True
```



## Exiting Data Entry

Entry mode can be terminated procedurally by using [AL\\_ExitCell](#).

## Commands

### AL\_SetEnterable

(areaRef:L; columnNumber:I; enterability:I; popupArray:X; menuPackRef:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column to apply enterability
→ enterability	integer	Enterability mode
→ popupArray	array	4D array to display in popup menu
→ menuPackRef	longint	Reference to a MenuPack menu

**AL\_SetEnterable** is used to set the enterability of a column.

**columnNumber** — This parameter specifies what column to act on. If **columnNumber** is 0, then all columns will be affected.

**enterability** — This parameter specifies the methods of enterability for **columnNumber**:

Value	Description
0	Not enterable
1	Enterable using typed characters only (default)
2	Enterable using popup menu only
3	Enterable using both typed characters and popup menu
4	Enterable using popup menu only (no meta characters)
5	Enterable using both typed characters and popup menu (no meta characters)

You can use **enterability** 4 or 5 to control meta characters for a given column. If you wish to define the meta character functions globally, use the routine [AL\\_SetInterface](#) (**metaOption** parameter).

When using the enterability values 4 and 5, you can override the default setting configured by **AL\_SetInterface** for a given area. Therefore, if you wish to configure AreaList Pro to globally disable meta characters for popup controls, you can do so using **AL\_SetInterface**, then enable for a given column using **AL\_SetEnterable**.

**popupArray** — Array, integer, longint, real, string or text. This array will be displayed in the popup menu and must be the same type as the array or field displayed in **columnNumber**.



## Enterability

If it is not the same type or if it has no elements, then a menu containing a single disabled menu item with the text “No items in this menu” will be displayed.

An array is not needed to display a time or date popup menu; built-in menus are provided.

Columns containing boolean or picture arrays can not contain popup menus.

---

***Do not dispose of the array in 4D until the popup is no longer needed.***

---

**menuPackRef** — This parameter passes the reference obtained from the MenuPack/PopupPack plug-in. Please refer to the “Using PopupPack Popup Control with AreaList Pro” and “Using PopupPack Offscreen Popup Control with AreaList Pro” sections in the MenuPack Developer Reference manual.

If this parameter is not passed, then the values in **popupArray** will be displayed in an AreaList Pro popup.

---

***When the user selects an item from the MenuPack menu, the entry finished callback method is run. In this callback the appropriate MenuPack commands must then be called to determine the user’s selection. Then the user’s selection must be placed in the array element corresponding to the cell entered.***

---

If this command is called in the On Plug in Area event phase or in the Event Callback Interface method, use [AL\\_UpdateArrays](#) or [AL\\_UpdateFields](#) to redraw the AreaList Pro object as needed.

See [Using Callback Methods During Data Entry](#) for a discussion of using callback methods with popup menus.

If this command is not called, then all columns will be enterable using typed characters only.

Examples:

**AL\_SetEnterable**(eList;4;1) `set column 4 to be enterable using typed characters only

**AL\_SetEnterable**(eList;0;0) `set all columns to be not enterable

**AL\_SetEnterable**(eList;3;2;aProducts) `set the third column to be enterable via a popup menu containing the items in the array aProducts

The following example will disable meta characters globally, for all AreaList Pro areas:

**AL\_SetInterface**(0;-1;-1;-1;1) `0 as area parameter defines for all AreaList Pro areas

Then, you can enable meta characters for a given column in a specific area:

**AL\_SetEnterable**(eList;2;2;atTest)

If the data type is time or date, [AL\\_SetInterface](#) is also used to specify the interface to be used:

- Whenever a cell is enterable with popup, the **useOldPopup** parameter of [AL\\_SetInterface](#) sets what kind of popup will be used (old or new). See [Data Entry Using Popups](#) for more details.
- Whenever a cell is enterable with typing, the **entryControls** parameter of [AL\\_SetInterface](#) sets if the entry is performed using plain text or inline controls. See [Data Entry Using Inline Controls](#) for more details.

These settings can be restricted (but not expanded) with [AL\\_SetCellEnter](#).

---

## AL\_SetFilter

(areaRef:L; columnNumber:I; entryFilter:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column where to apply filter
→ entryFilter	string	Filter for input data

**AL\_SetFilter** is used to set the entry filter for a column.

**columnNumber** — This parameter specifies the column to act on. If **columnNumber** is 0, then all columns will be affected.

**entryFilter** — This parameter specifies the filter to use. Entry filters will function as they do in 4<sup>th</sup> Dimension, except that they will not handle placeholders. Predefined styles may not be used.

Please read the section [Filters](#) for more information.

Examples:

**AL\_SetFilter**(eList;3;"&9") `column 3, allow numbers

**AL\_SetFilter**(eList;6;"~a") `column 6, allow lower and uppercase, make all uppercase

CapsFilter:="~"+**Char**(34)+"A-Z;a-z;0-9;.;0;/;\*;(;&;\$;\;" +**Char**(34)

**AL\_SetFilter**(eList;4;CapsFilter) `column 4, allow multiple groups and several individual characters

## AL\_SetEntryOpts

(areaRef:L; entryMode:I; allowReturn:I; displaySeconds:I; moveWithArrows:I; mapEnterKey:I; decimalCharForWin:S; useNewPopuIcon:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ entryMode	integer	Mode to initiate entry
→ allowReturn	string	Allow entry of carriage returns into text arrays
→ displaySeconds	integer	Display seconds in time arrays during data entry
→ moveWithArrows	integer	Move enterable cell using the Arrow keys
→ mapEnterKey	integer	Map the Enter key to function as another key
→ decimalCharForWin	string	Decimal character under 4D for Windows
→ useNewPopuIcon	integer	Display modern popup icon

**AL\_SetEntryOpts** is used to control several AreaList Pro options pertaining to data entry. Please read the section [Moving the Current Entry Cell](#) for more information.

**entryMode** — 0 to 7 (8 to 15 to ignore soft deselect). This option determines the mode that the user can use to initiate data entry and select rows with the mouse. The table below describes the possible values. The default is 1.

Value	Entry	Selection
0	None	Single-click
1	None	Single and double-click
2	Single-click	None
3	Double-click	Single-click
4	<ctrl/command> Double-click	Single and double-click
5	<shift>Double-click	Single and double-click
6	<option/alt> Double-click	Single and double-click
7	<control>Double-click	Single and double-click

**To ignore soft deselect events (e.g. clicking on non-focusable button) add 8 to the entryMode parameter.**

See the sections [Compatibility Note — New Menu Architecture](#) and [Compatibility Note — AL ExitCell and AL Cell deselect action become AL ExitCell and AL Cell Validate](#) for details about “hard deselect” and “soft deselect”.

The following example will configure area to ignore soft deselect events:

**AL\_SetEntryOpts**(eList;10;...)`single-click to initiate entry, no row selection, ignore soft deselect

## Enterability

**allowReturn** — 0 or 1:

- 0** — the Carriage Return key will move the enterable cell as described in the [Moving the Current Entry Cell](#) (default)
- 1** — the user can enter a carriage return character into a text array element

**displaySeconds** — 0 or 1:

- 0** — seconds will not be displayed (default)
- 1** — seconds will be displayed in time array elements during data entry

**moveWithArrows** — 0 or 1:

- 0** — the Arrow keys will move the insertion point within the enterable cell (default)
- 1** — the Arrow keys will move the enterable cell to the next cell according to the key pressed

**mapEnterKey** — The Enter key is in many cases used to accept a record or perform some other action in 4D. If the Enter key is not acting as expected, make sure that it is not being used as a key equivalent somewhere on the layout.

- 0** — do not map the Enter key (default)
- 1** — map the Enter key to act like the Tab key
- 2** — map the Enter key to act like the Return key

**decimalCharForWin** — Character to be interpreted as the decimal point. Only the first character of **decimalCharForWin** is used for the decimal point, any other characters will be ignored. The default is the US decimal point ".".

**decimalCharForWin** applies only to AreaList Pro running under 4D for Windows. On MacOS, when a real number is entered into a cell during data entry, MacOS converts the text entered into a real number (after exiting the cell). MacOS takes into account the decimal point set in the International preferences. This feature is not available under all versions of Windows.

**useNewPopuIcon** — 0 or 1:

- 0** — all popup icons will have the old black and white look (default)
- 1** — all popup icons will have a 3D look

Examples:

`Initiate data entry with a double-click, single-click selection, don't allow carriage return characters to be entered into text arrays, don't display seconds in time arrays during data entry, map the Enter key to act like the Tab key, modern popups

**AL\_SetEntryOpts**(eList;3;0;0;0;1;".";1)

`Initiate data entry with a single-click, no selection, allow carriage return characters to be entered into text arrays, display seconds in time arrays during data entry, use Arrows to navigate between cells, black and white popups

**AL\_SetEntryOpts**(eList;2;1;1;1;0;".";0)

## AL\_SetEntryCtrls

(areaRef:L; columnNumber:I; controlType:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	Column in which control appears
→ controlType	integer	Type of control

**AL\_SetEntryCtrls** is used to specify which type of control will be used for data entry in a column displaying a boolean array. If the column contains any other type of array, this command will be ignored.

**columnNumber** — This parameter specifies the column to act on.

**controlType** — 0 or 1:

**0** — checkbox without title (default)

**1** — checkbox with title (the title is the **True** label specified in [AL\\_SetFormat](#))

## AL\_SetCellEnter

(areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; enterability:I)

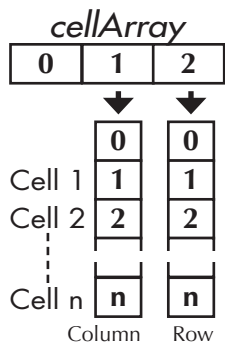
Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ firstCellCol	integer	First cell column
→ firstCellRow	longint	First cell row
→ lastCellCol	integer	Last cell column
→ lastCellRow	longint	Last cell row
→ cellArray	two-dimensional longint array	Discontiguous cells
→ enterability	integer	Enterability

**AL\_SetCellEnter** is used to set the enterability of a specific cell, range of cells, or list of cells.

- **To specify a single cell.** If **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** or **lastCellRow** are less than or equal to 0 then only [**firstCellCol**, **firstCellRow**] will be set.
- **To specify a range of cells.** If **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** and **lastCellRow** are greater than 0 then the range of cells from [**firstCellCol**, **firstCellRow**] to [**lastCellCol**, **lastCellRow**] will be set.
- **To specify discontiguous cells.** If **firstCellCol** or **firstCellRow** are less than or equal to 0 then the cells in **cellArray** will be set.

## Enterability

**cellArray** — Two-dimensional long integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.



**enterability** — 0, 1, 2, 3 or -1:

**0** — the cell is not enterable

**1** — the cell is enterable by typing or popup, according to the **enterability** parameter of [AL\\_SetEnterable](#) for the column

**2** — the cell is enterable by popup only, if the **enterability** parameter of [AL\\_SetEnterable](#) for the column is set accordingly

**3** — the cell is enterable by typing only, if the **enterability** parameter of [AL\\_SetEnterable](#) for the column is set accordingly

**-1** — remove any cell-specific enterability which has been set for the cells

The **moveWithData** option of [AL\\_SetCellOpts](#) controls whether cell enterability stays with a cell whenever sorting, row dragging, or column dragging occurs.

Regarding enterability by popup, the popup is displayed in the cell if the column property is set as such by [AL\\_SetEnterable](#), which defines if the column cells will be enterable with typing and/or popup.

If the column is set as enterable with typing and popup, before version 8.1 it was only possible to set which cells were enterable (with both methods) or not (with none of the two methods). Values 2 and 3 of the **enterability** parameter now allow to set cells that will be enterable with popup or typing only. This setting is only useful when the column property is enterable with both popup and typing.

- Whenever a cell is enterable with popup, the **useOldPopup** parameter of [AL\\_SetInterface](#) sets what kind of popup will be used (old or new). See [Data Entry Using Popups](#) for more details.
- Whenever a cell is enterable with typing, the **entryControls** parameter of [AL\\_SetInterface](#) sets if the entry is performed using plain text of inline controls. See [Data Entry Using Inline Controls](#) for more details.

## Enterability

Examples:

**ARRAY LONGINT**(aCellArray;2;0)

`Set the cell in the third column, first row, to be enterable

**AL\_SetCellEnter**(eList;3;1;0;0;aCellArray;1)

`Set the cells in the fourth row (ten columns) to be non-enterable

`the row number is the same for all the cells, just the column number changes: range of values

**AL\_SetCellEnter**(eList;1;4;10;4;aCellArray;0)

`Set the cells in rows 8, 9, and 10, the first two columns, to be non-enterable

**AL\_SetCellEnter**(eList;1;8;2;10;aCellArray;0)

---

## AL\_GetCellEnter

(areaRef:L; cellColumn:L; cellRow:L; enterability:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellColumn	integer	Cell column
→ cellRow	longint	Cell row
← enterability	integer	Enterability

**AL\_GetCellEnter** is used to determine if the enterability of the specified cell has been explicitly set with [AL\\_SetCellEnter](#).

---

**Note that AL\_GetCellEnter will not get the column enterability.**

---

enterability — 1, 0 or -1:

**1** — the cell is enterable by typing

**0** — the cell is not enterable by typing

**-1** — the cell's enterability has not been previously set for the cell

## AL\_GetCellMod

(areaRef:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← resultCode	longint	Result code

**AL\_GetCellMod** will report whether or not the contents of the cell have been modified.

Use this command in the **entryFinishedMethod** callback.

Please read the section [Executing a Callback Upon Leaving a Cell](#) for more information.

**resultCode** — This parameter reports whether or not a cell was modified:

**0** — not modified

**1** — modified

Example:

**If**(**AL\_GetCellMod**(eList)=1)`was the value modified?

**AL\_GetCurrCell**(eList;vCol;vRow)

**If**(vCol=5)`5th column is Line Item quantity

αExtended{vRow}:=αQty{vRow}\*αPrice{vRow}

**AL\_UpdateArrays**(eList;-1)

**End if**

**End if**

## AL\_GetCellValue

(areaRef:L; cellRow:L; cellColumn:I; alphanumericData:T; pictData:P)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellRow	longint	Desired row
→ cellColumn	longint	Desired column
← alphanumericData	text	Non picture data
← pictData	picture	Picture data

**AL\_GetCellValue** will return the displayed value using the supplied row and column.



## Enterability

If the value contained in the cell is a picture object, it will be returned in the picture data parameter, otherwise the value will be returned in the non-picture data parameter.

**cellRow** — Desired row.

**cellColumn** — Desired column.

**alphanumericData** — Any non-picture data will be returned in this parameter, using any formatting values if supplied (see [AL\\_SetFormat](#)).

**pictData** — Any picture data will be returned in this parameter.

The following example will extract the cell data for the 3<sup>rd</sup> row, 2<sup>nd</sup> column. The information returned will be either text or picture:

```
C_TEXT($tData)
```

```
C_PICTURE($pData)
```

```
AL_GetCellValue(eList;3;2;$tData;$pData)
```

---

## AL\_SetCellHigh

(areaRef:L; startPosition:I; endPosition:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ startPosition	integer	First character of cell text to highlight
→ endPosition	integer	Last character of cell text to highlight

**AL\_SetCellHigh** will highlight a range of characters within a cell, from **startPosition** to **endPosition-1**. When **startPosition** = **endPosition**, then the insertion point will be positioned prior to the character indicated in **startPosition**, and none of the characters in the cell will be highlighted.

Example:

```
`Entry finished callback:
```

```
If(Not(vDataValid))
```

```
    AL_SetCellHigh(eList;vStart;vEnd) `highlight the cell contents to indicate error
```

```
End if
```

## AL\_GetCellHigh

(areaRef:L; startPosition:I; endPosition:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← startPosition	integer	First character of highlighted cell text
← endPosition	integer	Last character of highlighted cell text

**AL\_GetCellHigh** will obtain the highlighted range of characters within a cell. This command may be used to provide user feedback after performing error checking on entered data, and can be used in the entry finished callback method. Please read the section [Executing a Callback Upon Leaving a Cell](#) for more information. See [AL\\_SetCellHigh](#).

**AL\_GetCellHigh** and **AL\_SetCellHigh** also work during data entry (cell editing).

**startPosition** — This parameter indicates the first highlighted character.

**endPosition** — This parameter indicates the last highlighted character.

## AL\_SetCellIcon

(areaRef:L; cellColumn:I; cellRow:L; pictRef:P; iconAlignment:I; horPosition:I; vertPosition:I; offset:I; scaling:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellColumn	integer	Column at which to set the icon
→ cellRow	longint	Row at which to set the icon
→ iconRef	longint	Reference of the icon or picture to use
→ iconAlignment	integer	Position of icon
→ horPosition	integer	Horizontal position
→ vertPosition	integer	Vertical position
→ offset	integer	Pixel offset
→ scaling	integer	Scaling

**AL\_SetCellIcon** provides the ability to procedurally place icons in individual cells.

One or two icons may be used (left and right). You can customize the icon(s) using “cicn” or “PICT” resources, or items from the 4D Picture Library (see details below).

---

*This call supersedes **Escape sentence icons placed in cells** (see [Header/Cell Icon Support](#)).*

---

**cellColumn** — Desired cell column number.

**cellRow** — Desired cell row number.

**iconRef** — Reference of the icon or picture to use. Both “cicn” and “PICT” resources can be used, as well as items from the Picture Library. To associate an icon to the cell, pass one of the following numeric values ([Use PICT resource](#) and [Use PicRef](#) are 4D constants):

- N, where N is the resource ID of Mac OS-based “cicn” resource
- [Use PICT resource](#) + N, where N is the the resource ID of a Mac OS-based “PICT” resource
- [Use PicRef](#) + N, where N is the reference number of a picture from the Design environment Picture Library
- pass zero (0) if you do not want any icon for the cell

See [Header/Cell Icon Support](#) for examples. See also the 4<sup>th</sup> Dimension Language Reference regarding the **SET LIST ITEM PROPERTIES** command, which uses the same icon syntax.

**iconAlignment** — Position of icon (each cell can contain up to two icons):

- 0 — places icon on left of cell
- 1 — places icon on right of cell

**horPosition** — One the following options:

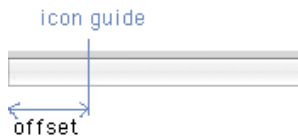
- 0 — default (left for left icon, right for right icon)
- 1 — align left
- 2 — align center
- 3 — align right

**vertPosition** — One the following options:

- 0 — default (top)
- 1 — align top left
- 2 — align center
- 3 — align bottom

**offset** — offset of the “icon guide”. The horizontal position is relative to this position. If the horizontal alignment is center, the icon is centered between the guide and corresponding side of cell (left for left icon, right for right icon).

The picture below illustrates the icon guide and its offset:



ICON GUIDE AND OFFSET

In the picture below, the left icon is aligned right to the icon guide and the right icon is aligned left to the icon guide:



LEFT ICON ALIGNED RIGHT

RIGHT ICON ALIGNED LEFT

In the picture below, the left icon is centered between the left border and the icon guide and no right icon is used:



LEFT ICON CENTERED

**scaling** — One the following options:

**0** — truncated

**1** — scaled

The cell content (text) is drawn into the space that is left once the icon is drawn. If the icon is larger than the remaining available space, the text is drawn over the icon.

For example, if the column width is 100 pixels and you draw a 15 pixel icon, there is remaining width of 85 pixels where the text will be drawn. If, however, the total width (icon + text) exceeds the column width, the text will be drawn over the picture. This allows background pictures behind the text.

The following example will draw an icon in r3c2, using an item (resID 1717) from the Picture Library:

```
$col:=2
$row:=3
$iconRef:=1717+Use PicRef
$iconPos:=1 `right
$horPos:=0 `default
$verPos:=2 `align center
$offset:=5
$scaling:=0
```

**AL\_SetCellIcon** (eAL\_Output; \$col; \$row; \$iconRef; \$iconPos; \$horPos; \$verPos; \$offset; \$scaling)

## AL\_GotoCell

(areaRef:L; cellColumn:I; cellRow:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ cellColumn	integer	Column to move entry to
→ cellRow	longint	Row to move entry to

**AL\_GotoCell** will place the cursor into the specified cell. If the cell does not exist or has been set to not enterable by [AL\\_SetEnterable](#), then this command will have no effect.

If you use **AL\_GotoCell** from an object method or project method other than the entry or exit callback method, you must precede it with the 4D command **GOTO AREA**. This is because **AL\_GotoCell** only works if the AreaList Pro object is selected.

If **AL\_GotoCell** is called in the On load phase, the AreaList Pro area must be the first in the entry order for the layout.

**cellColumn** — This parameter specifies the cell's column.

**cellRow** — This parameter specifies the cell's row.

Example:

```

`Entry callback method
AL_GetCurrCell(eltems;vCol;vRow)
If(vCol=3) `unit price
  If(gAccess#"Sales") `does user have security access to this field?
    If($2=2) `Tab
      AL_GotoCell(eltems;vCol+1;vRow) `goto the next cell
    Else `not Tab
      AL_ExitCell(eltems) `end data entry
    End if
  End if
End if

```

### AL\_GetCurrCell

(areaRef:L; cellColumn:I; cellRow:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← cellColumn	integer	Cell column
← cellRow	longint	Cell row

**AL\_GetCurrCell** will return the currently enterable cell. This command is only valid from a callback method. Please read the section [Using Callback Methods During Data Entry](#) for more information.

**cellColumn** — This parameter returns the current cell's column number.

**cellRow** — This parameter returns the current cell's row number.

**AL\_GetCurrCell** will return 0 in both **cellColumn** and **cellRow** if there is not a cell being entered.

Example:

```
AL_GetCurrCell(eList;vColumn;vRow) `get the current cell
```

### AL\_GetPrevCell

(areaRef:L; cellColumn:I; cellRow:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← cellColumn	integer	Column where entry cell was located
← cellRow	longint	Row where entry cell was located

**AL\_GetPrevCell** will return the previously enterable cell.

**cellColumn** — This parameter returns the previous cell's column number.

**cellRow** — This parameter returns the previous cell's row number.

## AL\_SkipCell

(areaRef:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout

**AL\_SkipCell** will skip the current data entry cell and proceed to the next cell.

This command can only be called from the entry started callback method. Please read the section [Executing a Callback Upon Entering a Cell](#) for more information.

If data entry in a cell is begun via a Tab, shift-Tab, Return, shift-Return, or click, then **AL\_SkipCell** moves data entry to the next appropriate cell, according to the entry method.

If the cell was entered via a mouse click, the cell will be exited and data entry will be ended.

If the cell was entered because of **AL\_SkipCell** called from the previous cell, then data entry will similarly be moved to the next cell.

If the method by which data entry begun is anything else, this command will be ignored.

Example:

```
`Entry Callback Method
AL_GetCurrCell(eltems;vCol;vRow)
If (vCol=3) `unit price
  If (gAccess#"Sales") `does user have security access to this field?
    AL_SkipCell(eltems) `goto the next cell or end data entry
  End if
End if
```

## AL\_ExitCell

(areaRef:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout

**AL\_ExitCell** will exit the currently enterable cell. If there is not a cell being entered then **AL\_ExitCell** will have no effect.

**AL\_ExitCell** does not need to be used to deselect a cell undergoing data entry if:

- a menu is selected
- another layout object is clicked
- the user clicks elsewhere on the AreaList Pro object.

These cases will all terminate data entry normally without the use of this command, and the cell will receive its normal exit callback.

**AL\_ExitCell** is required, however, to terminate data entry from an entry callback method.

---

**Warning: in an enterable area, the row containing the currently edited data must not be deleted. **AL\_ExitCell** must be called before the row (array element) is deleted.**

---

See also [Compatibility Note — New Menu Architecture](#) and [Compatibility Note — AL ExitCell and AL Cell deselect action become AL ExitCell and AL Cell Validate](#) for details about “hard deselect” and “soft deselect”.

Example:

```
`Entry callback method
`don't allow entry into cell at column 3, row 4
AL_GetCurrCell(eList;vCol;vRow)
If ((vCol=3) & (vRow=4))
    AL_ExitCell(eList)
End if
```



# Dragging Commands

## Background

### Technical Details of the Dragging Implementation

You must configure AreaList Pro to allow dragging out of and into an AreaList Pro area.

Commands provide the control necessary to allow dragging within an area, between two or more areas, and to not allow dragging between certain areas.

To allow dragging out of AreaList Pro, you must pass an access “code” for the type of data that is to be dragged. You must specify the type of data to allow to be dragged and at least one code to enable dragging, using [AL\\_SetDrgSrc](#). Up to ten codes can be passed.

Allowing many codes provides for more flexibility in enabling and disabling dragging between various areas. This will be explained in more depth later.

In order to allow dragging into AreaList Pro, you must pass an access “code” for the type of data that can be the destination of a drag. You must specify the type of data that can receive a drag, and at least one code to enable dragging, using [AL\\_SetDrgDst](#). AreaList Pro supports dragging to rows, columns, and cells. As with [AL\\_SetDrgSrc](#), up to ten codes can be passed for flexibility reasons.

To enable cell dragging, the `cellSelection` option of [AL\\_SetCellOpts](#) must be set to 1 or 2 (single cell selection or multiple cells selection is enabled).

To drag a cell out of an AreaList Pro object, set the `sourceDataType` parameter of [AL\\_SetDrgSrc](#) to 3 ([AL Drag cell data type](#)).

To drag data into an AreaList Pro object and drop it as a cell, set the `destDataType` parameter of [AL\\_SetDrgDst](#) to 3 ([AL Drag cell data type](#)).

You can control the behavior of row dragging use the `dragOntoRow` parameter of [AL\\_SetDrgOpts](#). The two types of behavior are dragging to insert between two rows, or dragging onto a row. When dragging onto rows, AreaList Pro will not automatically reorder displayed arrays.

### What are access “codes”?

The access codes that are passed in the [AL\\_SetDrgSrc](#) and [AL\\_SetDrgDst](#) commands are used to enable dragging between specific drag partners. These drag partners can be the same AreaList Pro area, different AreaList Pro areas, or different plug-in areas.

When a drag takes place, the drag sender communicates its access codes to the drag receiver. The drag receiver will compare the access codes of the sender to its own codes. If any of the codes match, the drag is allowed. This mechanism allows a number of combinations between several drag partners. The following is an example of enabling the dragging of a row within the same AreaList Pro area.

## Dragging Commands

Example:

```
`Enable drag events to rows within the this area  
vSelfStr:=String(eList) `creates a unique code that only allows dragging within this area  
AL_SetDrgSrc(eList;1;vSelfStr) `row data type for source  
AL_SetDrgDst(eList;1;vSelfStr) `row data type for destination
```

This example also shows how you can create a unique identifier that only enables dragging within the same AreaList Pro area.

---

***AreaList Pro will update the arrays and refresh the area if the drag is within the same area (row-to-row or column-to-column).***

---

### After a drag

When a row, column or cell is dragged out of AreaList Pro, the following information is available to you:

- notification that a drag occurred
- which row, column or cell was dragged (index in array)
- where the row, column or cell was dragged to (this area or another area)

When the drag is completed, the AreaList Pro event callback (or area's object method) will run.

If a drag occurred, a \$2 event code of -5 is returned to the callback method (or [AL\\_GetLastEvent](#) command, formerly **ALProEvt** variable) if a row was dragged, -7 if a column was dragged, or -8 if a cell was dragged (see [Determining the User's Action on an AreaList Pro Object](#)).

Then [AL\\_GetDrgSrcRow](#) or [AL\\_GetDrgSrcCol](#) may be used to get the row, column or cell that was dragged.

To determine which plug-in area was the destination of the drag, call [AL\\_GetDrgArea](#). This command returns the **areaRef** (a long integer) and the process ID of the destination area, which may be the same AreaList Pro area, another AreaList Pro area, or another plug-in area.

When dragging to another object, that object can either reside in the same window or on another window, which may require use of 4D's **CALL PROCESS** command to take action on the drag — when dragging to other objects, AreaList Pro is only providing a user interface to the drag, and notifying you, the developer, that the drag has occurred.

You are responsible for manipulating any arrays or other data structures.

When an AreaList Pro area is the destination of a drag, the following information is available to you:

- the type of data that was the recipient of the drag (row, column or cell)
- the row, column or cell that was dragged to

## Dragging Commands

You must use [AL\\_GetDrgDstTyp](#) to determine if the destination of the drag was a row, column or cell:

- if the destination was a row, [AL\\_GetDrgDstRow](#) may be used to determine the destination row
- if the destination was a column, [AL\\_GetDrgDstCol](#) may be used to determine the destination column
- if the destination was a cell, then both **AL\_GetDrgDstRow** and **AL\_GetDrgDstCol** are used to determine the destination cell

If the destination of the drag is an area on another window, then you must use 4D's **CALL PROCESS** command to communicate to the other process.

---

***AreaList Pro will update the arrays and refresh the area if the drag is within the same area (row-to-row or column-to-column).***

---

Row dragging is disabled when an AreaList Pro object is in cell selection mode. Use the **moveWithData** option of [AL\\_SetCellOpts](#) to keep the cell-specific information with a cell when a row or column is dragged to a new location or the list is sorted.

---

***When dragging cells, there will be no automatic updating of arrays, even if the source and the destination lists are the same.***

---

## AreaList Pro on Multi-Page Layouts

You can place an AreaList Pro area on layouts that contain multiple pages. If you've configured the area to accept a drag from another area, you must enable and disable the AreaList Pro area using [AL\\_SetDrgDst](#), depending on whether the area is on the current page.

If the page containing the AreaList Pro area is not the current page, call **AL\_SetDrgDst** with empty strings for the **dstCode** parameters. When the page becomes current, call **AL\_SetDrgDst** with the actual **dstCode** values you wish to allow.

Please read the section [Drag and Drop — Changing Form Pages](#) for more information.

---

***You should always disable an AreaList Pro area which is not on the current layout page.***

---

## Multiple Rows Dragging

To enable multiple rows dragging, the following options must all be set as follows:

- the **cellSelection** option of [AL\\_SetCellOpts](#) must be set to 0 (row selection is enabled)
- the **multiRows** option of [AL\\_SetRowOpts](#) must be set to 1 (multiple rows selection is enabled)
- the **multiRowDrag** option of [AL\\_SetDrgOpts](#) must be set to 1 to enable multiple rows dragging.

To get the rows that were dragged, use [AL\\_GetSelect](#).

---

***When dragging multiple rows, there will be no automatic updating of arrays, even if the source and the destination lists are the same.***

---

### Drag DataType

The `dataType` parameters represent the type of the drag for both the source and the destination. They are used in the commands [AL\\_SetDrgSrc](#), [AL\\_SetDrgDst](#), and [AL\\_GetDrgDstTyp](#). These are the possible values:

- 1 — row
- 2 — column
- 3 — cell

### Drop Area

AreaList Pro includes a [Drop Area](#) object, which can be used as a destination for dragged rows and columns.

## Commands

---

### AL\_SetDrgSrc

(areaRef:L; sourceDataType:I; srcCode1:S; ...; srcCode10:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ sourceDataType	integer	Type of item dragged
→ srcCode1; ...; srcCode10	string	Used to match drag partners

**AL\_SetDrgSrc** is used to enable dragging out of the AreaList Pro object referenced by **areaRef**, by setting the access codes for the source of the drag.

This command must be called before a drag is initiated (usually in the [On load](#) phase). Please read the section [What are access “codes”?](#) for more information.

**sourceDataType** — Possible values are:

- 1 — row
- 2 — column
- 3 — cell

**srcCode** — 15 characters. The **srcCode** can have any value, such as “RowDrag”, “ColDrag”, “DragToALP”, etc.; however, it is meant to match a code passed into a potential drag partner. The drag partner will be the destination/receiver of the drag.

That destination can be the same AreaList Pro area, a different AreaList Pro area, or another object.

## Dragging Commands

This code can be any value other than an empty string. Avoid using the strings “TEXT” or “PICT”.

AreaList Pro performs the following logic during the actual drag: when the drag takes place, the source codes that were given in `srcCode1`, `srcCode2`, etc. will be communicated to the receiver of the drag. If any of the codes match, the drag is enabled.

See [What are access “codes”?](#)

Example:

```
`Enable dragging a row within this area
```

```
vSelfStr:=String(eList) `creates a unique code that only allows dragging within this area
```

```
AL_SetDrgSrc(eList;1;vSelfStr) `row data type for source
```

```
AL_SetDrgDst(eList;1;vSelfStr) `row data type for destination
```

---

### AL\_SetDrgDst

(areaRef:L; destDataType:I; dstCode1:S; ...; dstCode10:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ destDataType	integer	Data type to be received
→ dstCode1; ...; dstCode10	string	Access code(s) to be received

**AL\_SetDrgDst** is used to enable dragging into the destination area, by setting the access codes. Please read the section [What are access “codes”?](#) for more information.

This command must be called before a drag has occurred.

The `areaRef` parameter must be the destination (receiver) area of a drag.

**destDataType** — Possible values are:

- 1 — row
- 2 — column
- 3 — cell

For the data type specified by **destDataType** (either row, column, or cell), you must specify at least one **dstCode** to enable receiving of that type.

**dstCode** — 15 characters. The **dstCode** can be any value (other than an empty string), such as “Row-Drag”, “ColDrag”, “ALPDrag”, “PartNum”, etc. Avoid using the strings “TEXT” or “PICT”. Pass an empty string to disable dragging.

The code should be the same as what is passed into a potential drag partner. The drag partner will be the source/sender of the drag. The source area can be the same AreaList Pro area, a different AreaList Pro area, or another plug-in object.

## Dragging Commands

AreaList Pro performs the following logic during the actual drag: when the drag takes place, the destination codes that were given in `dstCode1`, `dstCode2`, etc. are compared to the source codes communicated by the sender of the drag. If any of the codes match, the drag is enabled.

See [Technical Details of the Dragging Implementation](#).

---

***When AreaList Pro is placed on a page in a multi-page layout, be sure to disable dragging using this command when that page is not the currently shown page. Please read the section [AreaList Pro on Multi-Page Layouts](#) for more information.***

---

Example:

``Enable dragging a row within this area`

`vSelfStr:=String(eList) `creates a unique code that only allows dragging within this area`

`AL_SetDrgDst(eList;1;vSelfStr) `row type for destination`

---

## AL\_SetDrgOpts

(areaRef:L; dragRowWithOptKey:I; scrollAreaSize:I; multiRowDrag:I; dragOntoRow:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ dragRowWithOptKey	integer	Drag row using the option/alt key
→ scrollAreaSize	integer	Size of area that will activate scrolling
→ multiRowDrag	integer	Enable multiple rows dragging
→ dragOntoRow	integer	Drag rows onto or between rows

**AL\_SetDrgOpts** is used to set various options to be used with dragging.

Call this command before a drag.

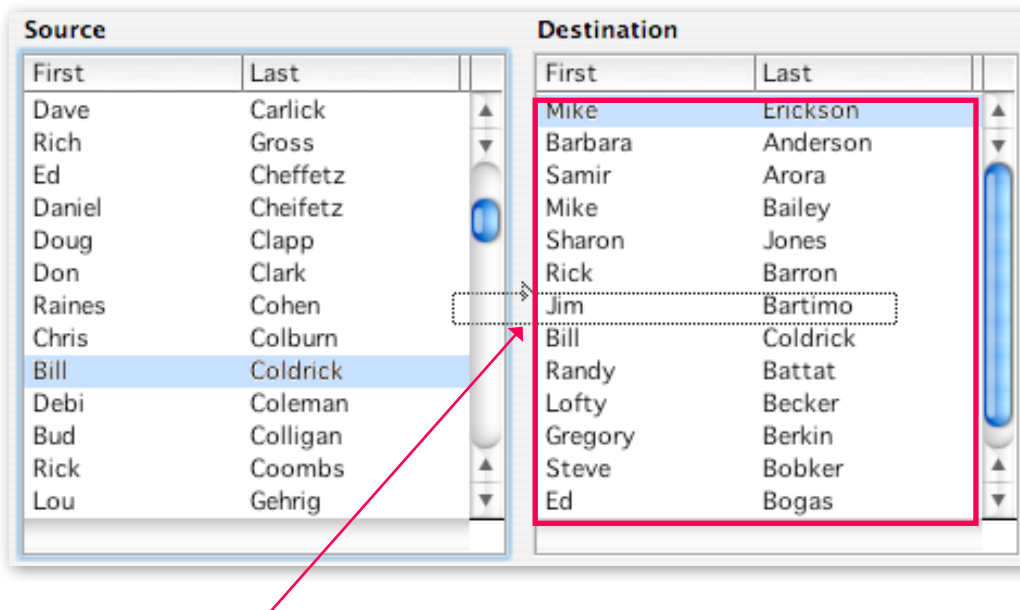
**dragRowWithOptKey** — 0 or 1:

**0** — the user can drag a row by clicking on it without holding down the option/alt key (default)

**1** — the user can drag a row by clicking on it while holding down the option/alt key

## Dragging Commands

**scrollAreaSize** — 0 to 30. This is the number of pixels outside of the destination area rectangle that will cause scrolling when the cursor is over it (see the illustration below). If **scrollAreaSize** is 0, then no scrolling will occur. The default is 30.



THE SCROLLAREASize IS CALCULATED FROM THIS DESTINATION AREA RECTANGLE WHEN RECEIVING A DRAG INTO A ROW

**multiRowDrag:**

- 1 — enable multiple rows dragging
- 0 — disable multiple rows dragging (default)

---

*With multiple rows dragging, the arrays or records will not be automatically updated even if the source and destination lists are the same. See [Multiple Rows Dragging](#) for more information.*

---

**dragOntoRow:**

- 1 — a row will be highlighted to receive the drag
- 0 — an insertion arrow will be displayed between rows (default)

---

*When dragging a row onto a row, there will be no automatic updating of arrays, even if the source and the destination lists are the same.*

---

Example:

`Drag row without the option key, scroll in 10 pixel area, drag multiple rows, drag between rows

**AL\_SetDrgOpts**(eList;0;10;1;0)

### AL\_GetDrgSrcRow

(areaRef:L; sourceRow:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← sourceRow	longint	Row that was dragged

Use **AL\_GetDrgSrcRow** to determine which row or cell was dragged after a drag has completed. The **areaRef** parameter should be the source (sender) area of a drag.

This command is called from the source area's object method when an event code of -5 (user dragged row) or -8 (user dragged cell) is returned to the callback method (or [AL\\_GetLastEvent](#) command, formerly **ALProEvt** variable).

See [Determining the User's Action on an AreaList Pro Object](#).

**sourceRow** — This parameter returns the row that was dragged.

Example:

```
`Event callback method
C_LONGINT($0) `object method and form method will not be executed if 0
C_LONGINT($1) `AreaList Pro area
C_LONGINT($2) `AreaList Pro event
C_LONGINT($3) `event modifier
C_LONGINT($4) `column — last clicked column
C_LONGINT($5) `row — last clicked row
C_LONGINT($6) `modifiers
C_STRING(255;$7) `tip string
C_STRING(255;$8) `area name
C_LONGINT(vRow)
Case of
:($2=-5) `user dragged a row
  AL_GetDrgSrcRow($1;vRow)
  `now do something useful
End case
```



### AL\_GetDrgSrcCol

(areaRef:L; sourceCol:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← sourceRow	integer	Column that was dragged

Use **AL\_GetDrgSrcCol** to determine which column or cell was dragged after a drag has completed. The **areaRef** parameter should be the source (sender) area of a drag. This command is called from the source area's object method when an event code of -7 (user dragged column) or -8 (user dragged cell) is returned to the callback method (or [AL\\_GetLastEvent](#) command (formerly **ALProEvt** variable). See [Determining the User's Action on an AreaList Pro Object](#).

**sourceCol** — This parameter returns the column that was dragged.

Example:

```
`Event callback method
C_LONGINT($0) `object method and form method will not be executed if 0
C_LONGINT($1) `AreaList Pro area
C_LONGINT($2) `AreaList Pro event
C_LONGINT($3) `event modifier
C_LONGINT($4) `column — last clicked column
C_LONGINT($5) `row — last clicked row
C_LONGINT($6) `modifiers
C_STRING(255;$7) `tip string
C_STRING(255;$8) `area name
C_LONGINT(vCol)
Case of
:($2=-7) `user dragged a column
  AL_GetDrgSrcCol($1;vCol)
  `now do something useful
End case
```

### AL\_GetDrgArea

(areaRef:L; destArea:L; destProcessID:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← destArea	longint	ID of the area the item was dragged to
← destProcessID	integer	Process ID of the destArea

Use **AL\_GetDrgArea** to determine the destination area of the last drag. The **areaRef** parameter should be the source (sender) area of a drag.

This command is called from the source area's object or form method (or event callback) when an event code of -5, -7, or -8 (user dragged a row, column or cell) is returned to the callback method (or by [AL\\_GetLastEvent](#) command, formerly **ALProEvt** variable).

See [Determining the User's Action on an AreaList Pro Object](#).

**destArea** — This parameter is the area reference of the area that is the destination of the drag.

**destProcessID** — This parameter contains the Process ID in which the window and destination area reside. Use the 4D command **CALL PROCESS** and the form event On Outside call for interprocess communication.

---

***If the destProcessID is different from the current process, you will need to use the 4D command CALL PROCESS and the form event On Outside call to communicate to the window that contains the destination area.***

---

Example:

```
`Event callback method
C_LONGINT($0) `object method and form method will not be executed if 0
C_LONGINT($1) `AreaList Pro area
C_LONGINT($2) `AreaList Pro event
C_LONGINT($3) `event modifier
C_LONGINT($4) `column — last clicked column
C_LONGINT($5) `row — last clicked row
C_LONGINT($6) `modifiers
C_STRING(255;$7) `tip string
C_STRING(255;$8) `area name
C_LONGINT(vDstArea;vDestID;vRow)
```

## Dragging Commands

### Case of

:( $\$2=5$ ) `user dragged a row

**AL\_GetDrgSrcRow**(\$1; vRow)

**AL\_GetDrgArea**(\$1;vDstArea;vDstID)

**If** (vDstID#**Current process**) `if dragged to a different process

◇vDstArea:=vDstArea `store in interprocess variable

**CALL PROCESS** (vDstID)

**End if**

**End case**

---

## AL\_GetDrgDstTyp

(areaRef:L; destDataType:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← destDataType	integer	Type of data which was destination of drag

**AL\_GetDrgDstTyp** is used to determine the type of data that was the destination of the last drag.

Specifically, the user may drag items to either a row, column, or a cell. After the drag has completed, **AL\_GetDrgDstTyp** indicates whether the destination of the drag was a row, column, or a cell.

The **areaRef** parameter should be the destination (receiver) area of a drag.

If the destination and source areas are actually the same area or different areas within the same process (i.e., they reside on the same layout), this command may be called from the source area's object or form method (or event callback).

If the destination and source areas are in different processes, then you will need to use the 4D command **CALL PROCESS** and the form event On Outside call and interprocess variables to communicate between the two processes.

**destDataType** — Indicates what type of data was the destination of the last drag:

- 1 — row
- 2 — column
- 3 — cell

## Dragging Commands

Example:

`Event callback method

**C\_LONGINT**(\$0) `object method and form method will not be executed if 0

**C\_LONGINT**(\$1) `AreaList Pro area

**C\_LONGINT**(\$2) `AreaList Pro event

**C\_LONGINT**(\$3) `event modifier

**C\_LONGINT**(\$4) `column — last clicked column

**C\_LONGINT**(\$5) `row — last clicked row

**C\_LONGINT**(\$6) `modifiers

**C\_STRING**(255;\$7) `tip string

**C\_STRING**(255;\$8) `area name

**C\_LONGINT**(vDstArea;vDestID;vDstType;vRow)

**Case of**

:(**\$2=-5**) `user dragged a row

**AL\_GetDrgSrcRow**(\$1; vRow)

**AL\_GetDrgArea**(\$1;vDstArea;vDestID)

**If**(vDstArea=\$1) `if dragged within the same area

**AL\_GetDrgDstTyp**(\$1;vDstType) `get the type of data that was destination of the drag

**If**(vDstTyp=1) `if dragged into a row

**AL\_GetDrgDstRow**(\$1;vRow) `get the row number

**End if**

**Else** `dragged to a different area

◇vDstArea:=vDstArea `store in interprocess variable

**CALL PROCESS**(vDestID)

**End if**

**End case**

`Destination AreaList Pro eDstALP area layout's form method

**C\_LONGINT**(vRow;vDstType)

**Case of**

:(**Form event= On Outside call**) `outside call (via CALL PROCESS)

**If**(◇vDstArea=eDstALP) `has a drag occurred from another process into this AreaList Pro object

**AL\_GetDrgDstTyp**(eDstALP;vDstType) `get the type of data that was destination of the drag

**If**(vDstTyp=1) `if dragged into a row

**AL\_GetDrgDstRow**(eDstALP;vRow) `get the row number

**End if**

**End if**

**End case**

### AL\_GetDrgDstRow

(areaRef:L; destRow:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← destRow	longint	Row number in area that was dragged to

If the destination of the last drag was a row or a cell (See [AL\\_GetDrgDstTyp](#)), use this command to determine which row or cell was the destination of the last drag. The **areaRef** parameter should be the destination (receiver) area of a drag.

If the destination and source areas are actually the same area or different areas within the same process (i.e., they reside on the same layout), this command may be called from the source area's object or form method (or event callback).

If the destination and source areas are in different processes, then you will need to use the 4D command **CALL PROCESS** and the form event On Outside call and interprocess variables to communicate between the two processes.

**destRow** — This parameter returns the row number of the destination area which received the drag.

See the [AL\\_GetDrgDstTyp](#) example above.

### AL\_GetDrgDstCol

(areaRef:L; destCol:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← destCol	integer	Column number in area that was dragged to

If the destination of the last drag was a column or a cell (See [AL\\_GetDrgDstTyp](#)), use this command to determine which column or cell was the destination of the last drag. The **areaRef** parameter should be the destination (receiver) area of a drag.

If the destination and source areas are actually the same area or different areas within the same process (i.e., they reside on the same layout), this command may be called from the source area's object or form method (or event callback).

If the destination and source areas are in different processes, then you will need to use the 4D command **CALL PROCESS** and the form event On Outside call and interprocess variables to communicate between the two processes.

**destCol** — This parameter returns the column number of the destination area which received the drag.

## Dragging Commands

Example:

```
`Event callback method
C_LONGINT($0) `object method and form method will not be executed if 0
C_LONGINT($1) `AreaList Pro area
C_LONGINT($2) `AreaList Pro event
C_LONGINT($3) `event modifier
C_LONGINT($4) `column — last clicked column
C_LONGINT($5) `row — last clicked row
C_LONGINT($6) `modifiers
C_STRING(255;$7) `tip string
C_STRING(255;$8) `area name
C_LONGINT(vDstArea;vDestID;vDstType;vCol)
Case of
:($2=-7) `user dragged a column
  AL_GetDrgSrcCol($1; vCol)
  AL_GetDrgArea($1;vDstArea;vDestID)
  If(vDstArea=$1) `if dragged within the same area
    AL_GetDrgDstTyp($1;vDstType) `get the type of data that was destination of the drag
    If(vDstTyp=2) `if dragged into a column
      AL_GetDrgDstCol($1;vCol) `get the column number
    End if
  Else `dragged to a different area
    ◇vDstArea:=vDstArea `store in interprocess variable
    CALL PROCESS(vDestID)
  End if
End case

`Destination AreaList Pro eDstALP area layout's form method
C_LONGINT(vCol;vDstType)
Case of
: (Form event= On Outside call) `outside call (via CALL PROCESS)
  If(◇vDstArea=eDstALP) `has a drag occurred from another process into this AreaList Pro object
    AL_GetDrgDstTyp(eDstALP;vDstType) `get the type of data that was destination of the drag
    If(vDstTyp=2) `if dragged into a column
      AL_GetDrgDstCol(eDstALP;vCol) `get the column number
    End if
  End if
End case
```

# User Action Commands

User interaction with an AreaList Pro object used to be handled in the **During** phase (On Plug in Area event) of the object method or form method, with the deprecated **postKey** paramter and **ALProEvt** process variable.

To accomplish this, you will most often use the various AreaList Pro commands from within this 4D method, which will also contain the response to user actions such as single-clicks and double-clicks.

A newer solution is the [AL\\_GetLastEvent](#) command and also the event callback project method set by [AL\\_SetEventCallback](#). See [Event Callback Interface](#) and [Event Callback vs Object Method](#).

## AreaList Pro's PostKey

In response to user activity, when using 4D 2003 and older versions, AreaList Pro had to send a message to 4D by posting a keyboard event to the event queue to trigger the object method or form method's execution in the **During** phase (On Plug in Area).

The default keyboard event is command/ctrl-\. You can modify this key with the **postKey** parameter of [AL\\_SetMiscOpts](#).

This is no longer the case with 4D 2004 and above.

## Determining the User's Action on an AreaList Pro Object

The type of user action that triggered the execution of the callback project method is returned by AreaList Pro to either of the two methods, according to the **flag** parameter of [AL\\_SetEventCallback](#):

- in the [AL\\_GetLastEvent](#) command (formerly **ALProEvt** process variable), to be used by the area's form or object method
- in the \$2 parameter of the event callback method

## User Action Commands

The possible values are:

Constant	Value	User Action
AL Null event	0	No action
AL Single click event	1	Single-click
AL Double click event	2	Double-click (see note below)
AL Empty Area Single click	3	Single-click in an empty part of the area (without displayed data)
AL Empty Area Double click	4	Double-click in an empty part of the area (without displayed data)
AL Single Control Click	5	Control-click (or right mouse click)
AL Empty Area Control Click	6	Control-click (or right mouse click) in an empty part of the area (without displayed data)
AL Vertical Scroll Event	7	Vertical scroll
AL Mouse moved event	18	Mouse moved (see note below)
AL Sort button event	-1	Sort button
AL Select all event	-2	Edit menu Select All
AL Column resize event	-3	Column resized
AL Column lock event	-4	Column lock changed
AL Row drag event	-5	Row dragged
AL Sort editor event	-6	Sort editor
AL Column drag event	-7	Column dragged
AL Cell drag event	-8	Cell dragged
AL Object resize event	-9	Object and window resized
AL Column click event	-10	User clicked on column header, automatic sort won't be executed
AL Column control click event	-11	Control-click on column header (see note below)
AL Footer click event	-12	Click on column footer

### Example

Typically, you will use the **If...End if** or **Case of...End** case commands to check the value of \$2/ [AL\\_GetLastEvent](#) command (formerly **ALProEvt** variable) .

If you had configured an AreaList Pro object to respond to both single and double-clicks, you might use a method like this in the event callback:

#### Case of

:\$2=1) `Single-click

:\$2=2) `Double-click

:\$2=3) `Single-click in an empty part of the area (without displayed data)

:\$2=4) `Double-click in an empty part of the area (without displayed data)

:\$2=5) `Control-click (or right mouse click)



## User Action Commands

:(\$2=6) `Control-click (or right mouse click) in an empty part of the area (without displayed data)

:(\$2=7) `Vertical scroll

:(\$2=18) `Mouse has been moved (callback method only)

:(\$2=-1) `Sort button

:(\$2=-2) `Edit menu Select All

:(\$2=-3) `Column resized

:(\$2=-4) `Column lock changed

:(\$2=-5) `Row has been dragged from this area

:(\$2=-6) `User has invoked AreaList Pro Sort Editor

:(\$2=-7) `Column has been dragged from this area

:(\$2=-8) `Cell has been dragged from this area

:(\$2=-9) `Object/window has been resized

:(\$2=-10) `User clicked on column header, automatic sort won't be executed

:(\$2=-11) `Control-click on column header

:(\$2=-12) `Click on column footer

**End case**

## Mouse Moved Event

18 (mouse moved) is only available in the Event Callback Interface.

It is NOT available through [AL\\_GetLastEvent](#) command (formerly **ALProEvt** variable).

## Single-click and Double-click Events

If a single-click is reported by AreaList Pro (event code 1), and the area is in single-row mode, you can determine whether the event was caused by a mouse click or by a keyboard event (the Arrow key or type-ahead scrolling). Both [AL\\_GetColumn](#) and [AL\\_GetClickedRow](#) will return zero if the event was due to an Arrow key or type-ahead scrolling.

A user double-click will not cause a call to the event callback method if the AreaList Pro object is configured to be enterable, and the selected data entry method is via a double-click.

If some of the columns are not enterable, a double-click on them will result in a single-click event.

Please read the section [Initiating Data Entry](#) for more information.

### Ctrl/command-click in the Column Header Event

The following actions will trigger a -11 event report without displaying the AreaList Pro Sort Editor:

- Windows right click
- MacOS ctrl-click
- MacOS right click

Windows ctrl-click and MacOS command-click won't report any event, but will trigger the display of the AreaList Pro Sort Editor if the value of `allowSortEditor` is set to 1 in [AL\\_SetSortOpts](#).

See the [section about this parameter](#).

### Event Callback vs Object Method

The `flag` parameter in [AL\\_SetEventCallback](#) and the `$0` result returned by the event callback project method determine if the object method (and form method) should be executed.

### Object Methods (or Project Methods) — On Plug in Area Event

When 4D executes an area object method or form method, neither 4D nor AreaList Pro is executing, which means that all AreaList Pro commands, all 4D commands and variables are available. The developer may do whatever he wishes. 4D will redraw the AreaList Pro area once the method has completed its execution. This is the secure and least problematic place where to place the code.

A limitation is that some events do not trigger the object method or project method execution. Another limitation is that no arguments (parameters) can be passed from or to the method.

### Event Callbacks

Area callbacks -event, entry and exit- are executed immediately after the event is received, often before this event is (fully) processed. They can receive parameters and modify AreaList Pro behavior.

However, when the callback is executed, 4D and AreaList Pro may be still processing the event, and some AreaList Pro and 4D commands are ignored. This is a more advanced feature and the developer should understand what he is doing. You need to watch carefully any modification of the data that is displayed by AreaList Pro. We try to ensure that at least AreaList Pro does not crash if the developer does something he should not, but some careless code can still get AreaList Pro into an "undefined state".

A good example is deleting row(s): if AreaList Pro has begun to process an event for a row and that row is then deleted in the callback method, once the execution returns from callback to AreaList Pro code, AreaList Pro proceeds with event processing, even though the row no longer exists. This may lead to an odd behavior.

### Using Both Methods

Event callback methods and object/form methods can be combined in specific cases. Communication between these code sets can be performed by saving the parameters passed to the event callback to 4D variables in the callback method, and then retrieve the values from these process variables in the area object method and/or the layout form method.

## Selection

You can determine what row or rows are selected using [AL\\_GetLine](#) if in single-row selection mode, and [AL\\_GetSelect](#) if in multiple rows selection mode.

If you are in cell selection mode, you can use [AL\\_GetCellSel](#) to determine the selected cells.

[AL\\_GetClickedRow](#) returns the last row that was clicked, which is different from **AL\_GetLine** (returns selected row).

## Sort Order

The user can change the sort order using the sort button (the column headers) or the Sort Editor. You can determine this sort order using [AL\\_GetSort](#).

## Column Widths

The user is able to resize the columns by clicking and dragging the dividing lines between columns. You can use [AL\\_GetWidths](#) to get the width of each column, in pixels.

## Column Information

[AL\\_GetColumn](#) is available to determine the column where a click occurred when selecting a row.

AreaList Pro allows one or more columns to be “locked” for horizontal scrolling.

If the `allowColumnLock` parameter of [AL\\_SetColOpts](#) is set, the user can change the column locked (see [Column Locking](#)).

You can determine the position of the column lock using [AL\\_GetColLock](#).

## Commands

### AL\_GetWidths

(areaRef:L; columnNumber:I; numWidths:I; width1:I; ...; widthN:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ columnNumber	integer	First column to get the width of
→ numWidths	integer	Number of widths to get (up to 15)
← width1; ...; widthN	integer	Pixel width of column

**AL\_GetWidths** is used to get the widths of the columns to allow any user changes to the column widths to be saved for future use. Up to fifteen widths can be retrieved at a time. Use [AL\\_SetWidths](#) to override the automatic column width sizing and set the widths of a column.

**columnNumber** — This parameter specifies the first column to get the width of.

**numWidths** — This parameter specifies the number of widths to get. This value should be equal to the number of variables passed for the **width** parameters.

**width1; ...; widthN** — These parameters return the pixel widths of the columns specified by **columnNumber** and **numWidths**.

Example:

#### Case of

: (Form event=On Load)

**SEARCH** ([Prefs];[Prefs]User=Current user)

\$error:=**AL\_SetArraysNam**(eList;1;4;"a1";"a2";"a3";"a4") `display the list

**AL\_SetWidths**(eList;1;4;[Prefs]Col1;[Prefs]Col2;[Prefs]Col3;[Prefs]Col4) `get previous widths

: (Form event=On Validate)

**AL\_GetWidths**(eList;1;4;vColumn1;vColumn2;vColumn3;vColumn4) `get the current widths

[Prefs]Col1:=vColumn1

[Prefs]Col2:=vColumn2

[Prefs]Col3:=vColumn3

[Prefs]Col4:=vColumn4

**SAVE RECORD** ([Prefs]) `save widths in a preferences file for future use

#### End case

## AL\_GetSort

(areaRef:L; column1:I; ...; columnN:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← column1; ...; columnN	integer	Columns that sort was performed upon

**AL\_GetSort** is used to return the current sort order.

**column1; ...; columnN** — These parameters return the column or columns that the user sorted. A **column** greater than 0 means that the column is sorted in ascending order, while a **column** less than 0 means that the column is sorted in descending order. If a **column** is 0 then all subsequent columns should be ignored.

When the user sort is bypassed by setting the **userSort** option of [AL\\_SetSortOpts](#) to 2, **AL\_GetSort** is still used to get the column header that was clicked on. You can set the sort order using [AL\\_SetSort](#).

Examples:

```

`Event callback method
C_LONGINT($0) `object method and form method will not be executed if 0
C_LONGINT($1) `AreaList Pro area
C_LONGINT($2) `AreaList Pro event
C_LONGINT($3) `event modifier
C_LONGINT($4) `column — last clicked columnNote
C_LONGINT($5) `row — last clicked row
C_LONGINT($6) `modifiers
C_STRING(255;$7) `tip string
C_STRING(255;$8) `area name
C_LONGINT(vSortCol;vCol1;vCol2;vCol3;vCol4;vCol5)
Case of
  :($2= -1) `user clicked a sort button
    AL_GetSort($1;vSortCol) `get the sorted column
End case
$Sorted:=AL_ShowSortEd($1) `display AreaList Pro Sort Editor
If($Sorted = 1)
  AL_GetSort($1;vCol1;vCol2;vCol3;vCol4;vCol5) `get the sort order
  `do something here
  AL_SetScroll($1;1;Abs(vCol1)) `scroll to the sorted column
End if

```

### AL\_GetSortedCols

(areaRef:L; sortList:X) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← sortList	longint array	Column order list
← resultCode	longint	Result code

**AL\_GetSortedCols** returns the current sort column numbers displayed in the Sort Editor.

You should use this routine after displaying the AreaList Pro Sort Editor.

By default, the Sort Editor will always use what the user has selected in the sort column and previous Sort Editor actions, unless you override the column list procedurally using [AL\\_SetSortedCols](#).

**sortList** — A valid 4<sup>th</sup> Dimension array (longint) which will receive the list of sort columns as displayed in the AreaList Pro Sort Editor.

- if the selected column was sorted in ascending order, the returned value will be positive
- if the selected column was in descending order, the returned value will be negative

The following example will retrieve the Sort Editor column list:

```
$ret:=AL_ShowSortEd(eList)
if ($ret=1)
    ARRAY LONGINT (aiAL_SortColList;0)
    $ret:=AL_GetSortCols(eList;aiAL_SortColList)
End if
```

### AL\_GetColumn

(areaRef:L) → clickedColumn:I

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← clickedColumn	integer	Column that the user clicked in

Use **AL\_GetColumn** to find out what column the user clicked in.

## User Action Commands

**clickedColumn** — This parameter returns the column that the user first clicked in (mouse-down). Thus if the user clicks in column 5 and then drags the mouse and releases it in column 8, the **clickedColumn** returned will be 5.

---

***AL\_GetColumn** and [AL\\_GetClickedRow](#) routines should not be used in entry or exit callback methods as it reports where the user has clicked, not where the cursor may reside. If you wish to get the current row and column within exit callback, you should use [AL\\_GetCurrCell](#).*

---

Example:

```
$column:=AL_GetColumn(eList) `get the column clicked on
```

---

*The column number is also available as parameter \$4 in the event callback method.*

---

## AL\_GetClickedRow

(areaRef:L) → clickedRow:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← clickedRow	longint	Row that the user clicked in

Use **AL\_GetClickedRow** (formerly **AL\_GetRow**) to find out what row the user clicked in.

This command should not be confused with [AL\\_GetLine](#). **AL\_GetClickedRow** returns the last row that was clicked, while **AL\_GetLine** returns the currently selected row, as a result of a click or any other action.

**clickedRow** — This parameter returns the row that the user first clicked in (mouse-down). Thus if the user clicks in row 5 and then drags the mouse and releases it in row 8, the **clickedRow** returned will be 5.

---

***AL\_GetColumn** and **AL\_GetClickedRow** routines should not be used in entry or exit callback methods as it reports where the user has clicked, not where the cursor may reside. If you wish to get the current row and column within exit callback, you should use [AL\\_GetCurrCell](#).*

---

## User Action Commands

Example:

```
$row:=AL_GetClickedRow(eList) `get the row clicked on
```

The following example will return the selected column and row numbers (cell), then use the [AL\\_GetCellValue](#) routine to return the associated data displayed in the selected cell, then use [AL\\_SetCellValue](#) to modify it:

### Case of

```
: (Form event=On Plug in Area)  
  $row:=AL_GetClickedRow(eList)  
  $col:=AL_GetColumn(eList)  
  $ret:=AL_GetCellValue(eList;$row;$col;$sData)  
  $sData:="new value"  
  AL_SetCellValue(eList;$row;$col;$sData)  
  AL_UpdateArrays(eList;-1)
```

### End case

---

*The row number is also available as parameter \$5 in the event callback method*

---

## AL\_GetSelect

(areaRef:L; array:X) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← array	longint array	Contains element numbers selected by the user when the multi-rows option is enabled
← resultCode	longint	Result code

**AL\_GetSelect** is used to determine which items were selected by the user when the **multiRows** option of [AL\\_SetRowOpts](#) is enabled, and they have selected multiple rows. Each element of the array contains a row number that the user selected when the list was displayed.

The array must be an long integer array, so be sure to use the **ARRAY LONGINT** command prior to calling **AL\_GetSelect**.

**resultCode** — This value is equal to one (1) if enough memory was available to resize **array**. If enough memory was not available you should react accordingly.



## User Action Commands

You can use [AL\\_SetSelect](#) to highlight rows.

Example: multi-rows option is enabled, the list is displayed, and the user selects rows 2,4,5,6,10,11,15,17,18,19,25.

`Area layout's form method

### Case of

: (**Form event**= On Load)

\$result:=**AL\_SetArraysNam**(eList;1;3;"aLN";"aFN";"aCompany") `display the list

**AL\_SetRowOpts**(eList;1;0;0;0;0) `turn on multi-rows option (2nd parameter)

\$result:=**AL\_SetEventCallback**(eList;"AL\_EventCallback";0) `do not execute area object or form method, update 4D variables

### End case

`AL\_EventCallback

`Event callback method

**C\_LONGINT**(\$0) `object method and form method will not be executed if 0

**C\_LONGINT**(\$1) `AreaList Pro area

**C\_LONGINT**(\$2) `AreaList Pro event

**C\_LONGINT**(\$3) `event modifier

**C\_LONGINT**(\$4) `column — last clicked column

**C\_LONGINT**(\$5) `row — last clicked row

**C\_LONGINT**(\$6) `modifiers

**C\_STRING**(255;\$7) `tip string

**C\_STRING**(255;\$8) `area name

**C\_LONGINT**(\$result;\$i)

### Case of

:(**\$2=1**)`user single-clicked

**ARRAY LONGINT**(aRows;0) `MUST use a long integer array!

\$result:=**AL\_GetSelect**(eList;aRows) `get the items selected by user

**If**(\$result=1)

**For**(\$i;1;**Size of array**(aRows)) `process each array item selected by user

**SEARCH**([Company];[Company]Name=aCompany{aRows{\$i}})

`do something here

### End for

**Else** `insufficient RAM to get user selection array

**ALERT**("Running low on memory, quit and restart!")

**End if** ` \$result=1

### End case

## AL\_GetCellSel

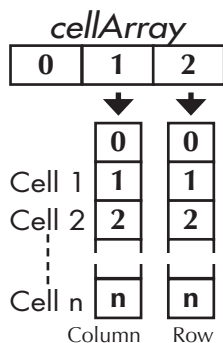
(areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X)  
→ resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← firstCellCol	integer	First cell column
← firstCellRow	longint	First cell row
← lastCellCol	integer	Last cell column
← lastCellRow	longint	Last cell row
← cellArray	two-dimensional longint array	Discontiguous cells
← resultCode	longint	Result code

**AL\_GetCellSel** is used to get the cell selection. Use the **cellSelection** option of [AL\\_SetCellOpts](#) to specify a cell selection mode prior to using this command. You can procedurally set the selected cells using [AL\\_SetCellSel](#).

- If only one cell is selected, then [firstCellCol, firstCellRow] will contain this cell and [lastCellCol, lastCellRow] will both be 0.
- If more than one cell is selected and all are contiguous, then [firstCellCol, firstCellRow] and [lastCellCol, lastCellRow] will contain the starting and ending points of this range.
- If more than one cell is selected but all are not contiguous, then [firstCellCol, firstCellRow] and [lastCellCol, lastCellRow] will all be 0 and **cellArray** will contain the selected cells.

**cellArray** — Two-dimensional long integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension will be set by AreaList Pro to be the same as the number of cells that are selected. See the following illustration.



**resultCode** — This value is equal to one (1) if enough memory was available to resize **cellArray**. If enough memory was not available you should react accordingly.

Example:

```
ARRAY LONGINT (aCellSelect;2;0)
```

```
AL_GetCellSel(eList;vFirstCol;vFirstRow;vLastCol;vLastRow;aCellSelect)
```

### AL\_GetScroll

(areaRef:L; verticalScroll:L; horizontalScroll:I)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← verticalScroll	longint	Vertical position (element #) list is scrolled to
← horizontalScroll	integer	Horizontal position (column #) list is scrolled to

**AL\_GetScroll** returns the current position of the thumb in the vertical and horizontal scrollbars.

**verticalScroll** — This parameter represents the element number visible at the top of the AreaList Pro display.

**horizontalScroll** — This parameter represents the column number visible at the left of the AreaList Pro display

The value returned in **horizontalScroll** represents the actual column number, including any columns which might be currently locked. For example, if the two left columns are locked, and the list is scrolled one column to the left, so that the fourth column is adjacent to the 2<sup>nd</sup> locked column, then the value returned is four.

You can set the scroll position using [AL\\_SetScroll](#).

Example:

```
AL_GetScroll(eList;vVert;vHoriz)
```

### AL\_GetColLock

(areaRef:L) → columns:I

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← columns	integer	Number of columns that are locked

**AL\_GetColLock** returns the number of columns currently locked.

**columns** — This parameter returns the number of columns currently locked.

You can set the lock position using [AL\\_SetColLock](#).

Example:

```
$lockcolumn:=AL_GetColLock(eList)
```

### AL\_GetLine

(areaRef:L) → selectedRow:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← selectedRow	longint	Number of currently selected row

**AL\_GetLine** returns the number of the currently selected row in the area specified by **areaRef**.

**AL\_GetLine** should only be used with an AreaList Pro object in single-row mode. If the object is in multi-rows mode, you should use [AL\\_GetSelect](#).

This command should not be confused with [AL\\_GetClickedRow](#). **AL\_GetClickedRow** returns the last row that was clicked, while **AL\_GetLine** returns the currently selected row, as a result of a click or any other action.

**selectedRow** — This parameter returns the number of the currently selected row.

You can set the selected row using [AL\\_SetLine](#).

Example:

```
`Modify button object method
`does a MODIFY RECORD on the record corresponding to the currently selected row in the
  AreaList Pro object eList
`uses an ID array (previously loaded from an ID field) to load the correct record
$row:=AL_GetLine(eList)
SEARCH ([Company];[Company]ID=aID{$row})
MODIFY RECORD ([Company])
```

### AL\_SetCellText

(areaRef:L; text:T; flag:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ text	text	Text you wish to set
→ flag	longint	Flag

**AL\_SetCellText** will set the currently highlighted cell text which can be obtained during Edit menu callback.

*This routine will only work correctly during Edit menu callback.*

## User Action Commands

**text** — Sets the current cell text based on flags.

**flags** — Formatting flags:

**0** — replace whole cell text

**1** — replace selected text only

The following line is called in the On Load phase:

```
$result:=AL_SetEditMenuCallback(area;"EditCallback")
```

The *EditCallback* project method is as follows:

```
C_LONGINT($1) `area reference
```

```
C_LONGINT($2) `event
```

```
C_TEXT($3;$AL_Undo) `unused
```

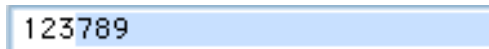
```
$text:="456"
```

```
AL_SetCellText($1;$text;1) `1 - replace selected text only
```

The user enters an editable cell:



Then a group of three characters is selected (highlighted):



The callback method is called at this moment, and the selected text is replaced:



---

## AL\_GetCellText

(areaRef:L; text:T; flag:L)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← text	text	Returns current cell highlight text
→ flag	longint	Flag

**AL\_GetCellText** will return the currently highlighted cell text which can be obtained during Edit menu callback.

---

***This routine will only work correctly during Edit menu callback.***

---

**text** — Returns current highlight text based on flags.

**flags** — Formatting flags:

**0** — get whole cell text

**1** — get selected text only

## User Action Commands

The following line is called in the On Load phase:

```
$result:=AL_SetEditMenuCallback(area;"EditCallback")
```

The *EditCallback* project method is as follows:

```
C_LONGINT($1) `area reference  
C_LONGINT($2) `event  
C_TEXT($3;$AL_Undo) `unused  
C_TEXT($text)  
AL_GetCellText($1;$text;1) `1 - get selected text only
```

The user enters an editable cell:



Then a group of three characters is selected (highlighted):



The callback method is called at this moment, with *\$text* containing the selected substring:

Expression	Value
 \$text	"amp"

---

## AL\_GetLastEvent

(areaRef:L) → eventCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout (optional)
← eventCode	longint	Event code

**AL\_GetLastEvent** will return the last event that occurred in the specified AreaList Pro area.

If no parameter is used, this command will return the last event that occurred in the last AreaList Pro area where an event occurred in the current process (it basically returns the old **ALProEvt** variable in this case).

Note that 18 (mouse moved) event is only available in the Event Callback Interface. It is NOT available through **AL\_GetLastEvent**.

This command replaces the deprecated **ALProEvt** variable (existing projects using this variable will still work, but **AL\_GetLastEvent** should be used instead in new projects or in case of issues with **ALProEvt**).

In addition, **AL\_GetLastEvent** offers a better control of event management compared to **ALProEvt**, as it allows tracking of the last event in each AreaList Pro area. Please refer to [Example 11](#).

# Utility Commands

AreaList Pro includes several commands to assist in managing the operation of an AreaList Pro area.

## Drop Area

AreaList Pro includes a simple plug-in area which functions as a “drop area” for rows, columns, or items dragged from other objects. Please read the section [Technical Details of the Dragging Implementation](#) for more information.

The Drop Area is essentially an invisible object which you will place on top of a graphic image (such as a trash can icon). You can control what types of objects can be dragged to the Drop Area using [AL\\_SetDropDst](#).

## Drop Area Objects on a Multi-Page Layout

If you are using a Drop Area on a layout with multiple pages, you must disable a Drop Area which is not on the active layout page. Use [AL\\_SetDropDst](#) with null strings for the `destCode` parameters.

## Disabling Drop Areas

You may want to disable a Drop Area so that it does not receive update events from 4D or the System. This involves the use of [AL\\_SetScroll\(area;0;0\)](#). See [Scroll bars — Changing Displayed Form](#).

## Sort Editor

AreaList Pro includes a Sort Editor dialog to allow the user to sort a list using more than one column as the sort criteria.

The user can command/ctrl-click on the headers to display the dialog. You can use [AL\\_ShowSortEd](#) to display this dialog procedurally.

## Area Name

[AL\\_SetAreaName](#) provides a more descriptive name for referencing AreaList Pro areas.

Pass a null string to clear the area name.

The name can also be obtained with [AL\\_GetAreaName](#). This area name is also available in the event callback method (parameter 8).

## Plug-in information

[AL\\_GetVersion](#) returns AreaList Pro’s version, while [AL\\_GetPluginPath](#) returns the active AreaList Pro copy location.

# Commands

---

## %AL\_DropArea

**%AL\_DropArea** is the command used to identify the plug-in area to which an AreaList Pro row or column can be dragged, but which does not display anything. When a row or column is dragged over this area, the area will invert.

This command will appear in the 4D Object Types popup on a layout Property List. It is only used in the object definition for an **%AL\_DropArea** object, and should never be used as a command in a 4D method.

---

## AL\_SetDropDst

(dropAreaRef:L; dstCode1:S; ...; dstCodeN:S)

Parameter	Type	Description
→ dropAreaRef	longint	Reference of Drop Area object on layout
→ dstCode1; ...; dstCodeN	string	String access code(s) to be received

**AL\_SetDropDst** is used to set the access codes for the destination of a drag, and should be called before a drag. Please read the section [“What are access “codes”?”](#) for more information.

**dstCode1; ...; dstCodeN** — 15 characters. The **dstCode** can be any value (other than an empty string), such as “RowDrag”, “ColDrag”, “ALPDrag”, “PartNum”, etc. Avoid using the strings “TEXT” or “PICT”.

The code should be the same as what is passed into a potential drag partner. The drag partner will be the source/sender of the drag. The source area can be an AreaList Pro area or another object.

The Drop Area performs the following logic during the actual drag: when the drag takes place, the destination codes that were given in **dstCode1**, **dstCode2**, etc. are compared to the source codes communicated by the sender of the drag. If any of the codes match, the drag is enabled.

Please read the section [Technical Details of the Dragging Implementation](#) for more information.

---

***When a Drop Area is placed on a page in a multi-page layout, be sure to disable dragging for that area by calling this command with null string for the dstCode parameters. Please read the section [Drop Area Objects on a Multi-Page Layout](#) for more information.***

---

Example:

```
`Enable dragging a row to this area
vStr:=String(eDrop) `creates a unique code that only allows dragging within this area
AL_SetDropDst(eList;vStr) `row type for destination
```



### AL\_ShowSortEd

(areaRef:L) → sortDone:I

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← sortDone	integer	User clicked the Sort button

**AL\_ShowSortEd** will display the AreaList Pro Sort Editor. The prompt may be set with the **sortEditorPrompt** parameter of [AL\\_SetSortOpts](#). The Editor will display the header values currently specified for the AreaList Pro object. The headers for picture columns will appear, but will be disabled.

Use [AL\\_GetSort](#) to determine what columns the user sorted on.

**sortDone** — This parameter returns what action the user made after the Sort Editor was displayed:

- 1 — The Sort Editor is being displayed in another process in that copy of 4D on that computer.  
In this case you can loop until a different value is returned or continue without sorting.
- 0 — The user clicked the Cancel button and the list was not sorted.
- 1 — The user clicked the Sort button and the list was sorted.

Example:

```
$sorted:=AL_ShowSortEd(eList) `display AreaList Pro Sort Editor
If($sorted = 1)
`do something here
End if
```

### AL\_SetAreaName

(areaRef:L; areaName:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
→ areaName	string	AreaList Pro area name

**AL\_SetAreaName** provides an interface for defining a name to a given AreaList Pro area. This can be helpful when using a generic event handler routine (see [AL\\_SetEventCallback](#)), providing a method for determining which area has passed along the desired event.

The following example will define the area name as “myArea” for the **eList** plug-in area:

```
AL_SetAreaName(eList,"myArea")
```

### AL\_GetAreaName

(areaRef:L; areaName:S)

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout
← areaName	string	AreaList Pro area name

**AL\_GetAreaName** returns the name defined by [AL\\_SetAreaName](#). This value is also available in the event callback method (parameter 8).

### AL\_GetVersion → version:S

Parameter	Type	Description
→ version	string	AreaList Pro version

**AL\_GetVersion** returns the current AreaList Pro's version.

Example:

```
$vers:=AL_GetVersion `returns the current plug-in version
```

### AL\_GetPluginPath → path:S

Parameter	Type	Description
→ path	string	Full plug-in pathname.

**AL\_GetPluginPath** returns the currently active AreaList Pro copy location on the hard drive, wherever it is located (including nested in the project folder of .bundle in the Plugins folder).

Example:

```
$path:=AL_GetPluginPath `returns the active plug-in path
```

# Obsolete Commands

Several commands are obsolete in AreaList Pro's current version, but are still supported for compatibility. You should not use these commands for new projects.

This chapter provides a simple list for these commands and their syntax.

**AL\_SetArrays** (areaRef:L; columnNumber:I; numArrays:I; array1:X; ...; arrayN:X) → resultCode:L

**AL\_InsertArrays** (areaRef:L; columnNumber:I; numArrays:I; array1:X; ...; arrayN:X) → resultCode:L

**AL\_SetForeClr** (areaRef:L; columnNumber:I; alpHdrForeColor:S; 4dHdrForeColor:I; alpListForeColor:S; 4dListForeColor:I; alpFtrForeColor:S; 4dFtrForeColor:I)

**AL\_SetBackClr** (areaRef:L; columnNumber:I; alpHdrBackColor:S; 4dHdrBackColor:I; alpListBackColor:S; 4dListBackColor:I; alpFtrBackColor:S; 4dFtrBackColor:I)

**AL\_DragMgrAvail** (isDragMgrPresent:I)

**AL\_GetDragLine** (areaRef:L; oldRowNumber:L; newRowNumber:L; destAreaName:L)

**AL\_GetDragCol** (areaRef:L; oldColumnNumber:I; newColumnNumber:I; destAreaName:L)

**AL\_SetDropOpts** (dropAreaRef:L; acceptRowDrag:I; acceptColumnDrag:I)

**AL\_SetWinLimits** (areaRef:L; enableResize:I; minWidth:I; minHeight:I; maxWidth:I; maxHeight:I)

**AL\_DoWinResize** (areaRef:L)

**AL\_SaveData** (areaRef:L; savePict:P) → resultCode:L

**AL\_RestoreData** (areaRef:L; restorePict:P) → resultCode:L

**AL\_SetSpellCheck** (areaRef:L; columnNumber:I; mode:I)

# Examples

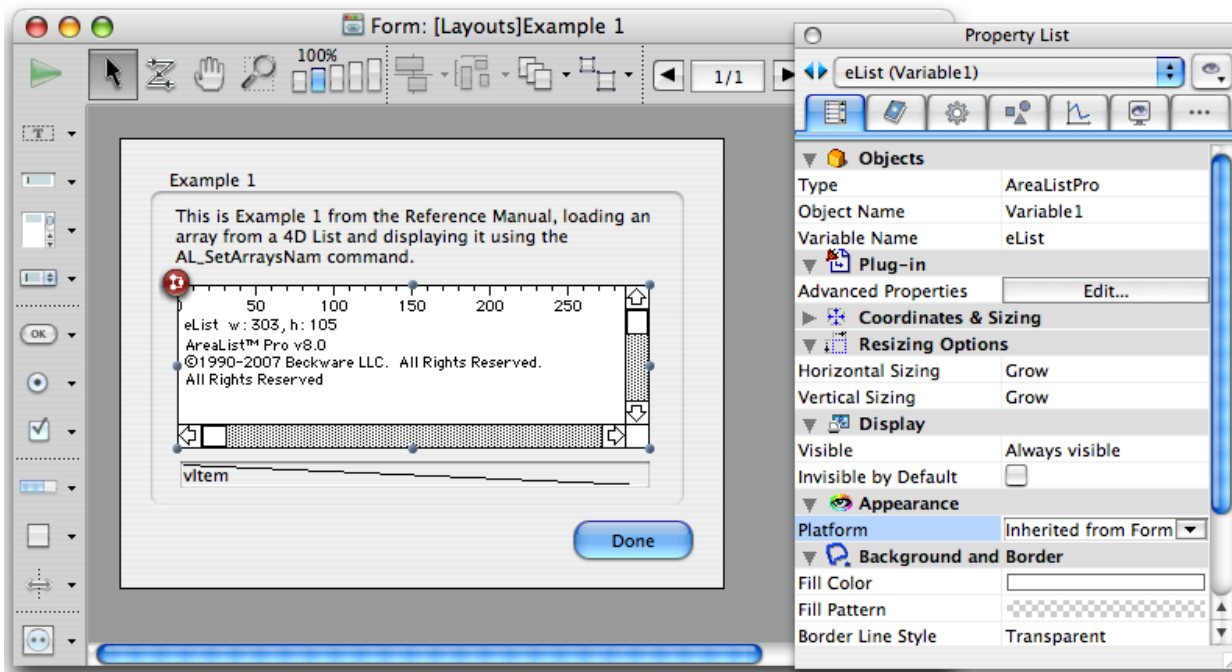
The examples in this section are designed to provide an overview of the use of AreaList Pro and the basic commands.

You may also wish to examine the non-compiled version of the AreaList Pro demo, for more examples on the various AreaList Pro capabilities.

## Example 1 — A Simple One-Column List

Create a simple one-column list on a 4D layout, containing the elements of a 4D list. When the user clicks on an element in the list, put the value of the selected element into a variable called `vItem`.

First we need to create the layout and draw the AreaList Pro plug-in object. We'll name the object `eList`.



## Examples

We'll use the On Plug in Area/[AL\\_GetLastEvent](#) command (formerly **ALProEvt** variable) system to configure our AreaList Pro area's response to user events.

Here is the **eList** object method:

### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
LIST TO ARRAY ("City, State";aCityState) `copy the list into an array
$errorcode:=AL_SetArraysNam(eList;1;1;"aCityState") `display array in AreaList Pro object
DEMO_Default(eList)
vItem:=aCityState{1}

: (Form event=On Plug in Area) `respond to user action
If (AL_GetLastEvent(eList)=1) `did user single-click on a row?
    $row:=AL_GetLine(eList) `get the row the user selected
    vItem:=aCityState{$row} `get the value in that element of the array
End if `AL_GetLastEvent(eList)=1
End case
```

The *DEMO\_Default* project method sets up the area's appearance according to the current platform:

```
C_LONGINT ($1;$AL_Area)
If (Count parameters>=1)
    $AL_Area:=$1
    AL_SetMiscOpts ($AL_Area;0;3;"";0;1)

If (IsWindows =1) `Windows
    AL_SetHdrStyle ($AL_Area;0;"Tahoma";11;0)
    AL_SetStyle ($AL_Area;0;"Tahoma";11;0)
Else `MacOS
    AL_SetStyle ($AL_Area;0;"Lucida Grande";11;0)
    AL_SetHdrStyle ($AL_Area;0;"Lucida Grande";11;0)
End if

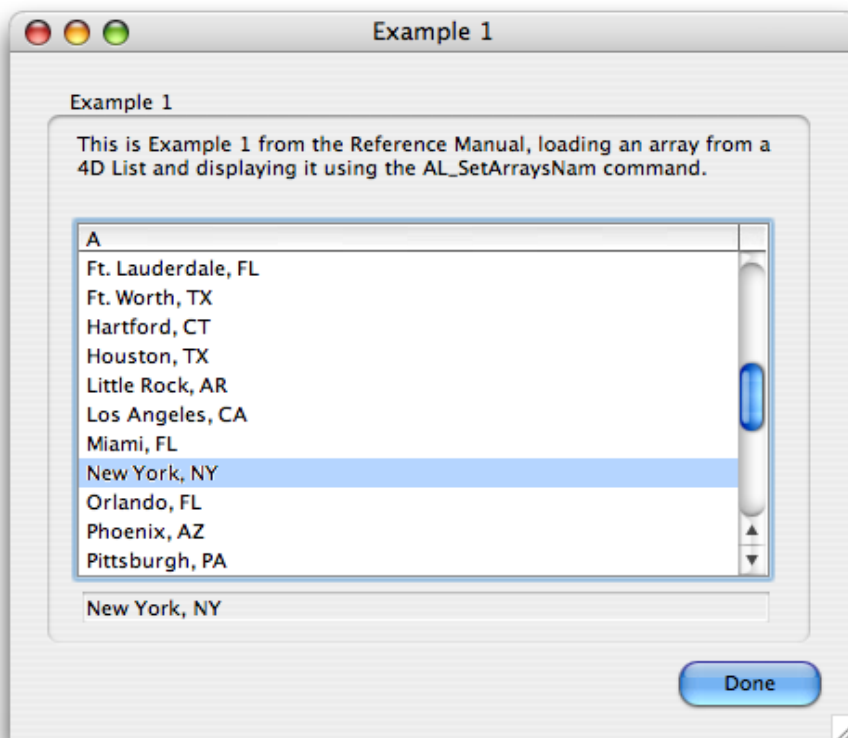
AL_SetHeight ($AL_Area;-1;-1;-1;3;-1;-1)
End if
```

## Examples

The *IsWindows* project method merely returns **True** if the current platform is Windows:

```
C_LONGINT($0;$platform;$system;$machine)
PLATFORM PROPERTIES($platform;$system;$machine)
$0:=Num($platform=Windows)
```

The layout will appear like this in the User or Runtime environment:



Notice that the column header displays the default value of "A". In the next example, we'll modify the display to have a more meaningful header.

### Example 2 — Displaying Headers on the List

Modify the previous example to display “City, State” as the header for the list column.

The modified object method for the AreaList Pro object is:

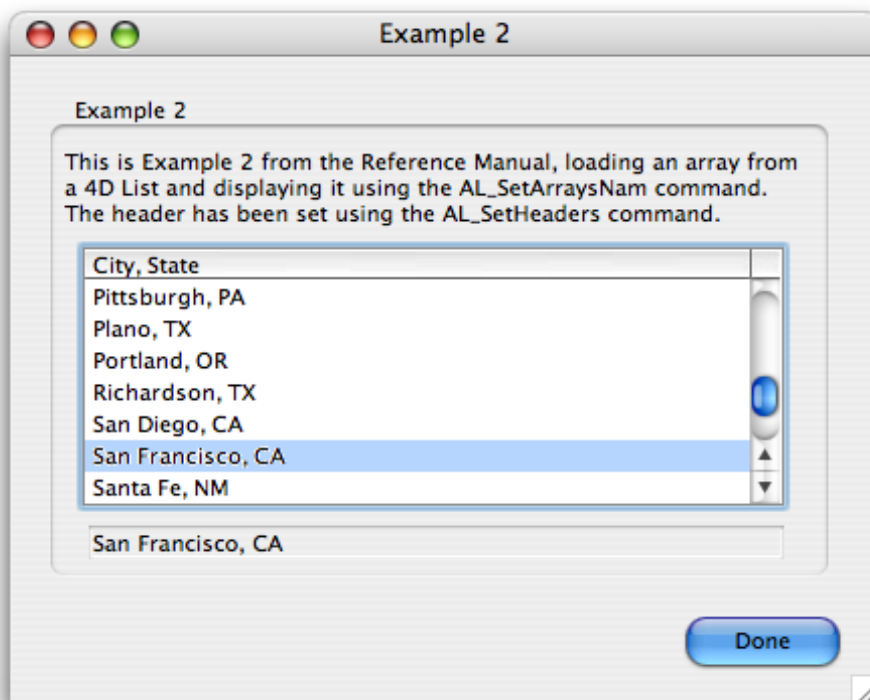
#### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object  
LIST TO ARRAY("City, State";aCityState) `copy the list into an array  
$errorcode:=AL_SetArraysNam(eList;1;1;"aCityState") `display array in AreaList Pro object  
AL_SetHeaders(eList;1;1;"City, State") `specify the values for the column header  
DEMO_Default(eList)  
vItem:=aCityState{1}
```

```
: (Form event=On Plug in Area) `respond to user action  
If(AL_GetLastEvent(eList)=1) `did user single-click on a row?  
$row:=AL_GetLine(eList) `get the row the user selected  
vItem:=aCityState{$row} `get the value in that element of the array  
End if `AL_GetLastEvent(eList)=1
```

#### End case

The AreaList Pro object now appears in the User or Runtime environment like this:



### Example 3 — Displaying Data from a Table

We'll change the previous example to load the array from a table in the database rather than a list. Tables are commonly used to keep list items when the number of items is large or may change frequently. Also, we'll display the City and State in separate columns.

This will require that our table structure keep the City and State values in two different fields. We can use the same AreaList Pro object we created in the previous examples, and just modify the object method:

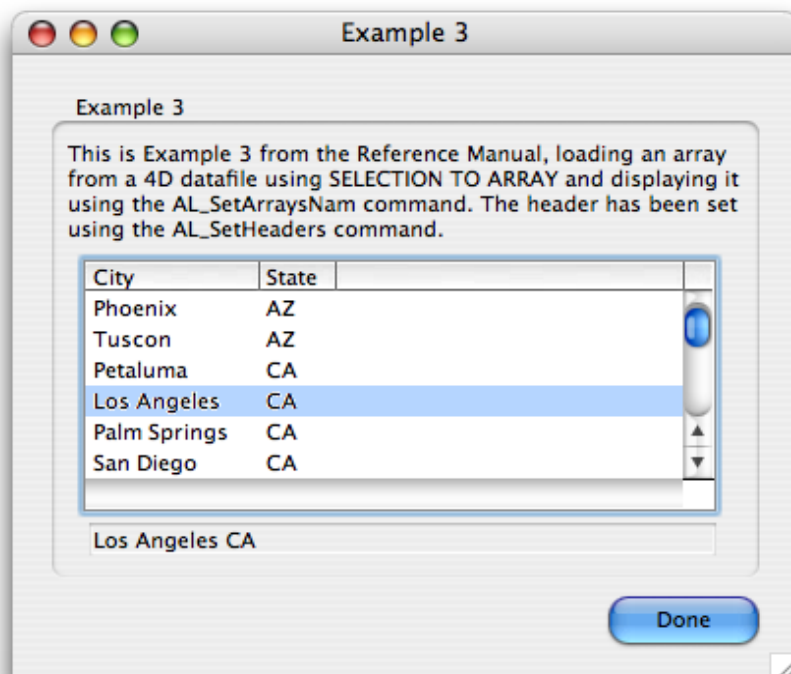
#### Case of

```
:(Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS[[Cities]] `load all records in the Cities table
SELECTION TO ARRAY[[Cities]City;aCity;[Cities]State;aState] `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
DEMO_Default(eList)
vItem:=aCity{1}+" ", "+aState{1}

:(Form event=On Plug in Area) `respond to user action
If(AL_GetLastEvent(eList)=1) `did user single-click on a row?
    $row:=AL_GetLine(eList) `get the row the user selected
    vItem:=aCity{$row}+" " +aState{$row} `get the value in that element of the arrays
End if `AL_GetLastEvent(eList)=1
```

#### End case

Our layout now looks like this to the user:





## Example 4 — Selecting Multiple Rows

In the previous examples, we've used the default single-row selection mode, which allows only one row to be selected, or highlighted, at any time. AreaList Pro can be configured to allow multiple rows to be selected, and commands are available to highlight rows procedurally, as well as determine what rows have been selected by the user.

Let's modify the previous example to work in multi-rows mode.

We'll add an additional line of code to the On load part of the object method to configure the AreaList Pro object to be in multi-rows mode, using [AL\\_SetRowOpts](#). We'll initially display the list with no rows selected. Since AreaList Pro defaults to no selected rows when in multi-rows mode, we don't need to use [AL\\_SetSelect](#) in the On load phase.

When the user clicks on one or more items, we'll display the selected items in the **vltem** variable, separated by a dash character. Finally, if the user double-clicks on a row, we want to close the layout using the **CANCEL** command.

Here's the modified object method:

### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS ([Cities]) `load all records in the Cities table
SELECTION TO ARRAY ([Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(eList;1;1;0;0) `set multi-rows mode and allow no selection parameters
DEMO_Default(eList)
vltem:=""
```

```
: (Form event=On Plug in Area) `respond to user action
```

### Case of

```
: (AL_GetLastEvent(eList)=1) `did user single-click on a row?
ARRAY LONGINT (aRows;0)
$OK:=AL_GetSelect(eList;aRows) `get the rows selected by user
vltem:=""
For ($i;1;Size of array (aRows)) `look at each row selected by user
    vltem:=vltem+aCity{aRows{$i}}+" "+aState{aRows{$i}}+" - " `plug values in vltem
End for
```

```
: (AL_GetLastEvent(eList)=2) `double-click?
```

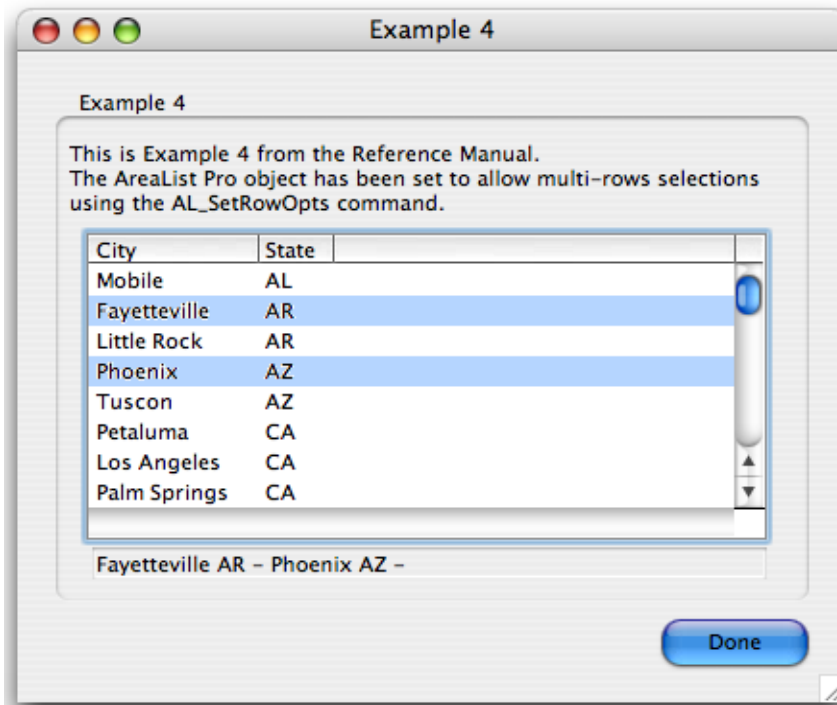
```
CANCEL `cancel the layout
```

```
End case `AL_GetLastEvent(eList)
```

```
End case
```

## Examples

Now our layout looks like this:



## Example 5 — Allowing Data Entry

Now that we have a basic AreaList Pro area displayed on our layout, we can implement data entry.

In AreaList Pro, all that needs to be done is to add a line of code to the On load part of the object method. To initiate data entry with a double click, we use [AL\\_SetEntryOpts](#) with the **entryMode** parameter set to 3 (see [Initiating Data Entry](#) for more information about the different options available).

As a default, AreaList Pro allows all columns to be enterable once the method of initiating data entry has been set.

Of course we must remove the double click **CANCEL** action. The code in the object method is now:

### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS ([Cities]) `load all records in the Cities table
SELECTION TO ARRAY ([Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(eList;1;1;0;0) `set multi-rows mode and allow no selection parameters
AL_SetEntryOpts(eList;3;0;0) `set double click to enter data entry mode
DEMO_Default(eList)
vItem:=""
```

## Examples

: (**Form event**=On Plug in Area) `respond to user action

**Case of**

: (**AL\_GetLastEvent**(eList)=1) `did user single-click on a row?

**ARRAY LONGINT** (aRows;0)

\$OK:=**AL\_GetSelect**(eList;aRows) `get the rows selected by user

vltem:=""

**For** (\$i;1;**Size of array** (aRows)) `look at each row selected by user

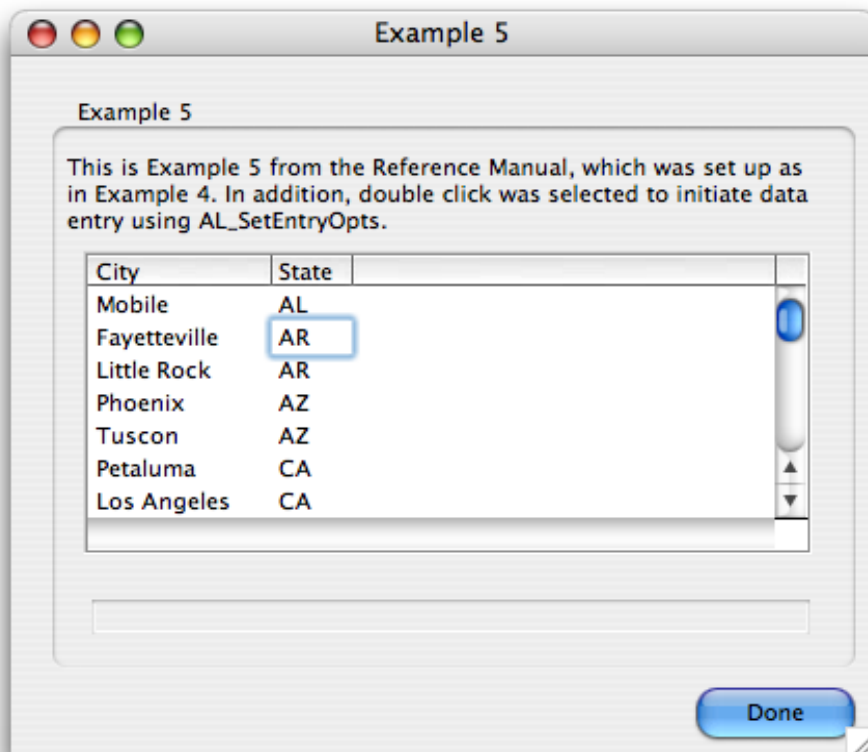
vltem:=vltem+aCity{aRows{\$i}}+" "+aState{aRows{\$i}}+" - " `plug values in vltem

**End for**

**End case** `AL\_GetLastEvent

**End case**

The layout now looks like this after double-clicking on the first cell with "AR" in it:



### Example 6 — Restricting Data Entry to a Column

Now that data entry has been established in our example AreaList Pro area, let's prohibit entry to one of the columns. This requires executing [AL\\_SetEnterable](#) to override the default enterability for column 1. In this command, which is also placed in the On load phase, we must specify the **columnNumber**, which is 1, and the **enterability**, which we'll set to 0 (not enterable.)

The modified object method is:

#### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS ([Cities]) `load all records in the Cities table
SELECTION TO ARRAY ([Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(eList;1;1;0;0) `set multi-rows mode and allow no selection parameters
AL_SetEntryOpts(eList;3;0;0) `set double click to enter data entry mode
AL_SetEnterable (eList;1;0) `set column 1 to be non-enterable
DEMO_Default(eList)
vItem:=""
```

```
: (Form event=On Plug in Area) `respond to user action
```

#### Case of

```
: (AL_GetLastEvent(eList)=1) `did user single-click on a row?
ARRAY LONGINT (aRows;0)
$OK:=AL_GetSelect(eList;aRows) `get the rows selected by user
vItem:=""
For ($i;1;Size of array (aRows)) `look at each row selected by user
    vItem:=vItem+aCity{aRows{$i}}+" "+aState{aRows{$i}}+" - " `plug values in vItem
End for
End case `AL_GetLastEvent (eList)
End case
```

This layout looks identical to that in Example 5, except that column 1 is no longer enterable. Test this by double clicking on column 1: the row will be selected, but you won't begin data entry.

Double clicking on column 2 will initiate data entry as in Example 5.

### Example 7 — Validating Data Entry

AreaList Pro has the capability to execute a 4<sup>th</sup> Dimension project method when data entry ends on a cell. This is known as a callback method, and can be specified using [AL\\_SetCallbacks](#).

In this example, we add a callback method which checks the value entered in a column 2 cell, and warns the user if it is invalid. To implement this, **AL\_SetCallbacks** is called from the On load phase, and sets up the callback project method *ExitCallback*.

The new object method is:

#### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS ([Cities]) `load all records in the Cities table
SELECTION TO ARRAY ([Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(eList;1;1;0;0) `set multi-rows mode and allow no selection parameters
AL_SetEntryOpts(eList;3;0;0) `set double click to enter data entry mode
AL_SetEnterable (eList;1;0) `set column 1 to be non-enterable
AL_SetCallbacks (eList;"";"ExitCallback") `set exit callback to project method ExitCallback
DEMO_Default(eList)
vItem:=""
```

```
: (Form event=On Plug in Area) `respond to user action
```

#### Case of

```
: (AL_GetLastEvent(eList)=1) `did user single-click on a row?
ARRAY LONGINT (aRows;0)
$OK:=AL_GetSelect(eList;aRows) `get the rows selected by user
vItem:=""
For ($i;1;Size of array (aRows)) `look at each row selected by user
    vItem:=vItem+aCity{aRows{$i}}+" "+aState{aRows{$i}}+" - " `plug values in vItem
End for
End case `AL_GetLastEvent (eList)
```

#### End case

In the callback method, we must find out from AreaList Pro if the cell was actually modified, and if so, which cell it was. [AL\\_GetCellMod](#) returns a boolean value indicating whether the cell was modified, and [AL\\_GetCurrCell](#) returns its column and row position.

Note that the callback method is actually a function. AreaList Pro expects a return value which will indicate whether or not the newly entered data is accepted.

## Examples

The code for the callback project method *ExitCallback* is:

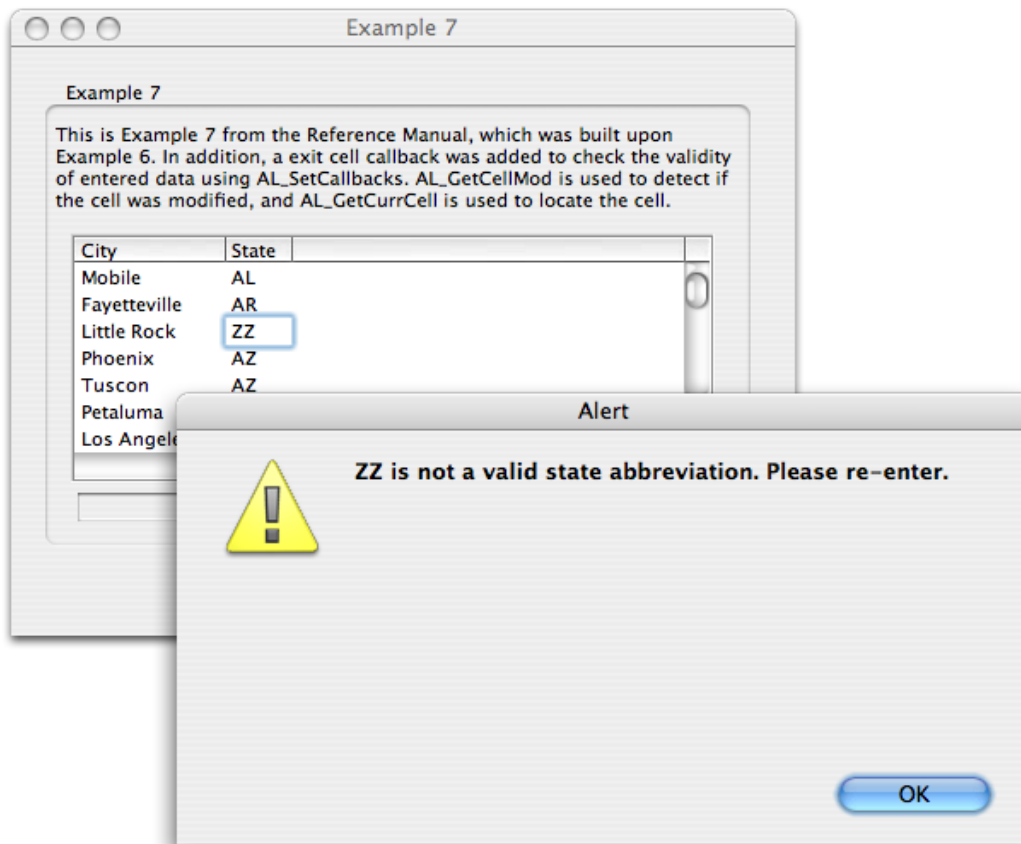
```
`Project method: ExitCallback
C_BOOLEAN($0) `"data valid" return value (True or False)
C_LONGINT($1) `AreaList Pro object reference
C_LONGINT($2) `action terminating data entry for this cell
C_LONGINT(vColumn;vRow)

If(AL_GetCellMod(eList)>0) `ask AreaList Pro if cell was modified
  AL_GetCurrCell(eList;vColumn;vRow) `find out which cell
  `since we only have one enterable array, we don't need to worry about the column
  LIST TO ARRAY("State Abbrev";aPossStates) `create a new array of all possible States
  If(Find in array(aPossStates;aState{vRow})=-1) `is modified element not valid?
    $0:=False `tell AreaList Pro it is invalid — this forces the user to re-enter it
    BEEP `provide user feedback
    ALERT(aState{vRow}+" is not a valid state abbreviation. Please re-enter.")
  Else
    $0:=True `tell AreaList Pro entry is valid
  End if

Else
  $0:=True `tell AreaList Pro entry is valid
End if
```

## Examples

In this layout, if the user wants to double click into the “State” column second cell with “AR” in it, enter “ZZ”, and exit the cell, this alert would be displayed:



## Example 8 — Prohibiting Data Entry to a Specific Cell

This example takes advantage of another possible AreaList Pro callback, which is executed when a cell is entered for data entry.

We'll make column 1 data (City) enterable again (as in Example 5) but we will use this callback to prohibit changes to for the state of California (abbreviation CA). The only other change necessary to the AreaList Pro object method is to add the entry callback project method name, *EntryCallback*, to the call to [AL\\_SetCallbacks](#).

The object method now is:

### Case of

: (**Form event=On Load**) `initialize the AreaList Pro object

**ALL RECORDS** ([Cities]) `load all records in the Cities table

**SELECTION TO ARRAY** ([Cities]City;aCity;[Cities]State;aState) `copy field values into arrays

## Examples

```
$errorCode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(eList;1;1;0;0) `set multi-rows mode and allow no selection parameters
AL_SetEntryOpts(eList;3;0;0) `set double click to enter data entry mode
AL_SetCallbacks (eList;"EntryCallback";"ExitCallback") `set callback project methods
DEMO_Default(eList)
vItem:=""
```

: (**Form event=On Plug in Area**) `respond to user action

**Case of**

: (**AL\_GetLastEvent**(eList)=1) `did user single-click on a row?

**ARRAY LONGINT** (aRows;0)

\$OK:=**AL\_GetSelect**(eList;aRows) `get the rows selected by user

vItem:=""

**For** (\$i;1;**Size of array** (aRows)) `look at each row selected by user

vItem:=vItem+aCity{aRows{\$i}}+" "+aState{aRows{\$i}}+" - " `plug values in vItem

**End for**

**End case** `AL\_GetLastEvent (eList)

**End case**

The entry callback method code checks the column number and row information by making use of [AL\\_GetCurrCell](#). Note that this method now makes use of the two parameters that AreaList Pro passes to it: the AreaList Pro area reference, and the method by which data entry was initiated.

`Project method: EntryCallback

**C\_LONGINT** (\$1) `AreaList Pro object reference

**C\_LONGINT** (\$2) `entry cause - passed by AreaList Pro

**C\_LONGINT** (\$3) `only useful when fields are being displayed

**C\_LONGINT** (vCurrCol;vCurrRow)

**AL\_GetCurrCell** (\$1;vCurrCol;vCurrRow)

**If** (vCurrCol=1) `city

**If** (aState{vCurrRow}="CA")

**AL\_SkipCell** (\$1)

**End if**

**End if**

When it is displayed in the Runtime environment, the layout in this example will look the same as the layout in Example 7. However, the cities in the first column are now enterable, except for those which are in California.



### Example 9 — Using the Event Callback Interface

This example show how a more generic event callback project method can be installed to replace the On Plug in Area/[AL\\_GetLastEvent](#) command (formerly **ALProEvt** variable) system.

This is performed by a call to [AL\\_SetEventCallback](#), which instructs AreaList Pro to call the *EventCallBack* project method instead of sending the On Plug in Area event to the object method and form method.

The area's object method is now only used for the On Load phase:

#### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS ([[Cities]]) `load all records in the Cities table
SELECTION TO ARRAY ([[Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(eList;1;1;0;0) `set multi-rows mode and allow no selection parameters
AL_SetEntryOpts(eList;3;0;0) `set double click to enter data entry mode
AL_SetCallbacks (eList;"EntryCallback";"ExitCallback") `set callback project methods
$errorcode:=AL_SetEventCallback (eList;"EventCallBack";3) `set event callback
` (3 = do not execute object method and form method)
DEMO_Default(eList)
vItem:=""
```

#### End case

The callback method code updates the **vItem** variable by making use of [AL\\_GetSelect](#). Note that this method makes use of the two first parameters that AreaList Pro passes to it: the AreaList Pro area reference, and the event which triggered the callback method execution.

The method also uses AreaList Pro's built-in constants:

```
`Project method: EventCallBack
C_LONGINT ($0) `object method and form method will not be executed if 0
C_LONGINT ($1;$area) `AreaList Pro area
C_LONGINT ($2;$alpEvent) `AreaList Pro event
C_LONGINT ($3;$alpEventMod) `event modifier — unused now
C_LONGINT ($4;$col) `column — last clicked column
C_LONGINT ($5;$row) `row — last clicked row
C_LONGINT ($6;$modifiers) `modifiers
C_STRING (255;$7;$tip) `tip string
C_STRING (255;$8;$areaName) `plug-in area name (see AL_SetAreaName)
```

## Examples

### Case of

: ((\$2>=AL Single click event) & (\$2<=AL Empty Area Control Click)) `all click types

**ARRAY LONGINT** (aRows;0)

\$OK:=**AL\_GetSelect**(eList;aRows) `get the rows selected by user

vItem:=""

**For** (\$i;1;**Size of array** (aRows)) `look at each row selected by user

vItem:=vItem+aCity{aRows{\$i}}+" "+aState{aRows{\$i}}+" - " `plug values in vItem

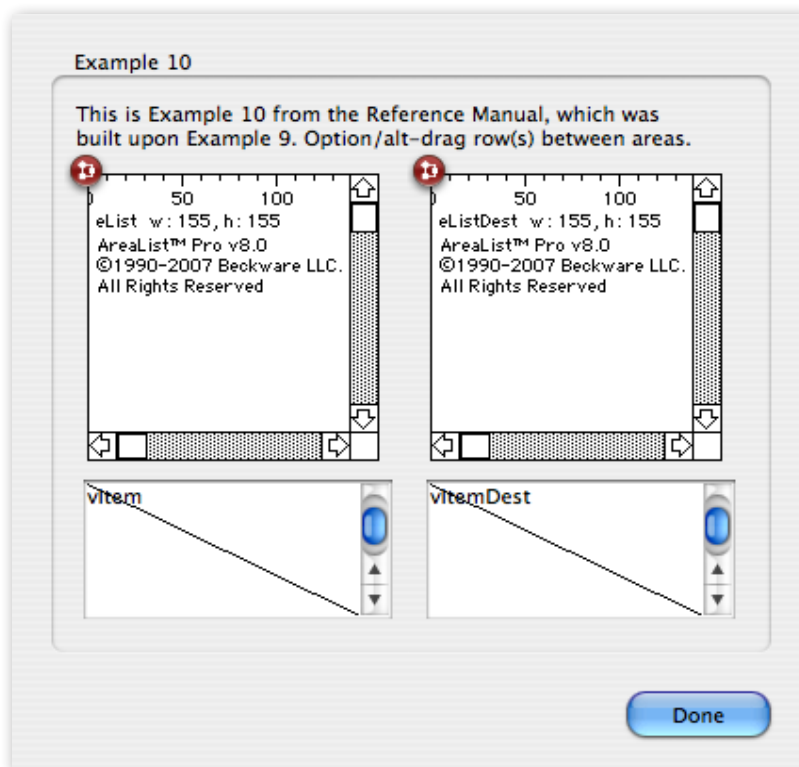
**End for**

**End case**

\$0:=0 `event handled

## Example 10 — Drag and Drop Between Areas

This example will demonstrate how to implement drag and drop between two AreaList Pro areas. Another area has been added to the right, with its corresponding variable to display selected row(s):



### Enabling Drag and Drop

The left area's object method allows multiple rows drag and drop to and from the area when the option/alt key is pressed. The "drag" access code is set for both areas.

Note that this code makes extensive use of AreaList Pro's built-in constants:

#### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS ([Cities]) `load all records in the Cities table
SELECTION TO ARRAY ([Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(eList;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(eList;AL Multiple row selection;AL Allow single or no row;2;1) `set multi-rows
mode, allow no selection, drag out, drag in
AL_SetDrgSrc(eList;AL Drag row data type;"drag") `set source access code
AL_SetDrgDst(eList;AL Drag row data type;"drag") `set destination access code
AL_SetDrgOpts(eList;AL Drag row with option key;AL Scroll area size default;
AL Multiple row dragging;AL Drag between rows) `multiple rows dragging with option key
AL_SetEntryOpts(eList;AL Entry dbl select sql;0;0) `set double click to enter data entry mode
AL_SetCallbacks (eList;"EntryCallback";"ExitCallback") `set callback project methods
$errorcode:=AL_SetEventCallback (eList;"EventCallBack";2) `set event callback
` (2 = do not execute object method and form method)
DEMO_Default(eList)
vItem:=""
```

#### End case

### Introducing Generic Programming

The destination area will behave the same way, except that it is empty until a drag occurs. This is where we want to go generic. First, we will call a common *dragAreaSetup* project method, with the area reference as \$1 parameter:

```
`dragAreaSetup
`called by both areas On Load phases
C_LONGINT($1) `AreaList Pro area
AL_SetHeaders($1;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts($1;AL Multiple row selection;AL Allow single or no row;2;1) `set multi-rows
mode, allow no selection, drag out, drag in
AL_SetDrgSrc($1;AL Drag row data type;"drag") `set source access code
AL_SetDrgDst($1;AL Drag row data type;"drag") `set destination access code
AL_SetDrgOpts($1;AL Drag row with option key;AL Scroll area size default;
AL Multiple row dragging;AL Drag between rows) `multiple rows dragging with option key
AL_SetEntryOpts($1;AL Entry dbl select sql;0;0) `set double click to enter data entry mode
AL_SetCallbacks($1;"EntryCallback";"ExitCallback") `set callback project methods
$errorcode:=AL_SetEventCallback($1;"EventCallBack";2) `set event callback
`{2 = do not execute object method and form method)
DEMO_Default($1)
```

Now our eList object method looks like this:

#### Case of

```
:(Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS[[Cities]] `load all records in the Cities table
SELECTION TO ARRAY[[Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(eList;1;2;"aCity";"aState") `display arrays in AreaList Pro object
dragAreaSetup(Self->)
vItem:=""
```

#### End case

The right area's object method does the same, but with empty arrays to begin with:

#### Case of

```
:(Form event=On Load) `initialize the AreaList Pro object
ARRAY TEXT(aCityDest;0)
ARRAY TEXT(aStateDest;0)
$errorcode:=AL_SetArraysNam(eList;1;2;"aCityDest";"aStateDest") `empty arrays in this area
dragAreaSetup(Self->)
vItemDest:=""
```

#### End case

### Updating Area Entry and Exit Callback Methods

We need to update our area entry and exit callbacks to make them compatible with both areas, using [AL\\_GetArrayNames](#):

```
`Project method: EntryCallback
C_LONGINT($1) `AreaList Pro object reference
C_LONGINT($2) `entry cause - passed by AreaList Pro
C_LONGINT($3) `only useful when fields are being displayed
C_LONGINT(vCurrCol;vCurrRow)
AL_GetCurrCell($1;vCurrCol;vCurrRow)
ARRAY TEXT(aArrayNames;0)
$errorcode:=AL_GetArrayNames($1;aArrayNames;0)
$ptrToStateArray:=Get pointer(aArrayNames{2}) `pointer to second col array (state)
If(vCurrCol=1) `city
    If($ptrToStateArray->{vCurrRow}="CA")
        AL_SkipCell($1)
    End if
End if

`Project method: ExitCallback
C_BOOLEAN($0) `"data valid" return value (True or False)
C_LONGINT($1) `AreaList Pro object reference
C_LONGINT($2) `action terminating data entry for this cell
C_LONGINT(vColumn;vRow)
If(AL_GetCellMod(eList)>0) `ask AreaList Pro if cell was modified
    AL_GetCurrCell(eList;vColumn;vRow) `find out which cell
    `only the state array col 2 will be checked, we don't need to worry about the entered column
    LIST TO ARRAY("State Abbrev";aPossStates) `create a new array of all possible States
    ARRAY TEXT(aArrayNames;0)
    $errorcode:=AL_GetArrayNames($1;aArrayNames;0)
    $ptrToStateArray:=Get pointer(aArrayNames{2}) `pointer to second col array (state)
    If(Find in array(aPossStates;$ptrToStateArray->{vRow})=-1) `is modified element not valid?
        $0:=False `tell AreaList Pro it is invalid — this forces the user to re-enter it
        BEEP `provide user feedback
        ALERT(aState{vRow}+" is not a valid state abbreviation. Please re-enter.")
    Else
        $0:=True `tell AreaList Pro entry is valid
    End if
Else
    $0:=True `tell AreaList Pro entry is valid
End if
```

## Examples

### Event Callback

First we create a common *evtUpdateText* project method to update either variable at the bottom of the lists. It assumes that the **aRows** array containing the selected rows has been updated prior to the method's call:

```
`evtUpdateText
```

```
`updates the variable at the bottom of the list
```

```
C_POINTER($1) `-> variable
```

```
C_POINTER($2) `-> city array
```

```
C_POINTER($3) `-> state array
```

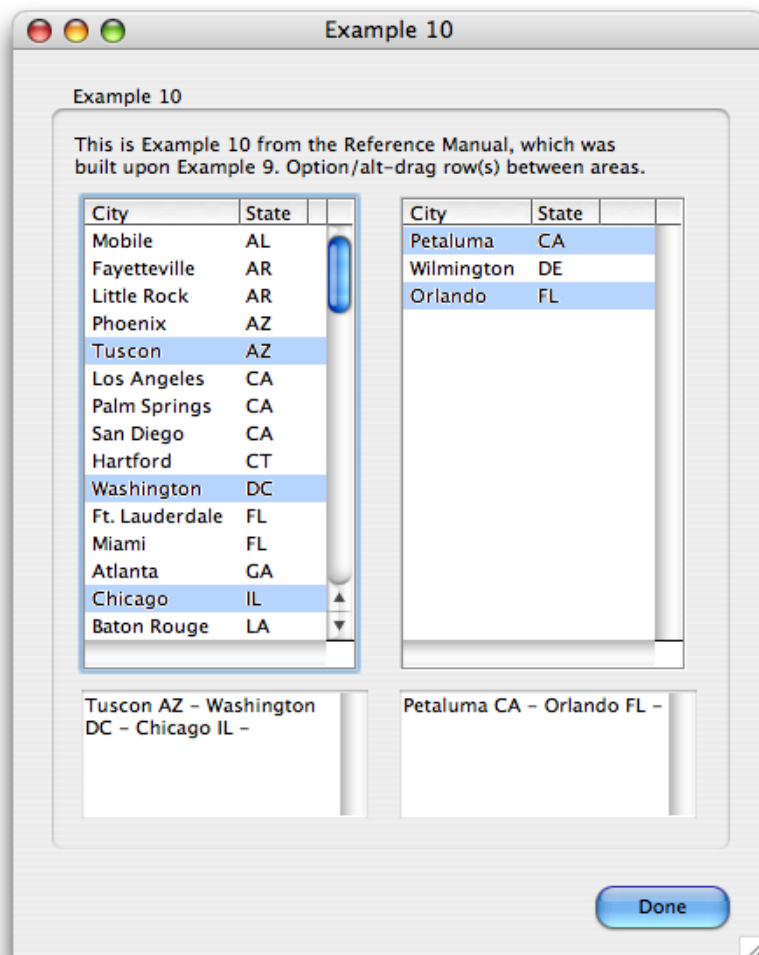
```
$1->:=""
```

```
For($i;1;Size of array(aRows)) `look at each row selected by user (aRows populated by event callback)
```

```
$1->:=$1->+$2->{aRows{$i}}+" "+$3->{aRows{$i}}+" - " `plug values in vItem or vItemDest
```

```
End for
```

Here is how the layout will look with a few rows selected in both areas, once rows have been dragged from the left area to the right one:



## Examples

The event callback method is common to both areas. It now calls *evtUpdateText* when a click occurs (or select all) and calls another project method named *evtRowsDragged* to handle a drag from the current (\$1) area:

`Project method: EventCallBack

**C\_LONGINT**(\$0) `object method and form method will not be executed if 0

**C\_LONGINT**(\$1;\$area) `AreaList Pro area

**C\_LONGINT**(\$2;\$alpEvent) `AreaList Pro event

**C\_LONGINT**(\$3;\$alpEventMod) `event modifier — unused now

**C\_LONGINT**(\$4;\$col) `column — last clicked column

**C\_LONGINT**(\$5;\$row) `row — last clicked row

**C\_LONGINT**(\$6;\$modifiers) `modifiers

**C\_STRING**(255;\$7;\$tip) `tip string

**C\_STRING**(255;\$8;\$areaName) `plug-in area name (see AL\_SetAreaName)

**ARRAY LONGINT**(aRows;0)

\$OK:=**AL\_GetSelect**(eList;aRows) `get the rows selected by user

### Case of

: ((((\$2=>AL Single click event) & (\$2<=<u>AL Empty Area Control Click</u>)) | (\$2=<u>AL Select all event</u>))

`all click types or select all

**If**(\$1=eList) `left area

*evtUpdateText*(->vItem;->aCity;->aState)

**Else** `right area

*evtUpdateText*(->vItemDest;->aCityDest;->aStateDest)

**End if**

: (\$2=<u>AL Row drag event</u>) `row(s) dragged from this area

**If**(\$1=eList) `source is left area

*evtRowsDragged*(\$1;->aCity;->aState;->aCityDest;->aStateDest)

vItem:= "" `nothing selected on the left (rows have been removed)

*evtUpdateText*(->vItemDest;->aCityDest;->aStateDest) `update variable for right area with the new rows

**AL\_SetWidths**(eListDest;1;2;0;0) `resize columns in destination area

**Else** `source is right area

*evtRowsDragged*(\$1;->aCityDest;->aStateDest;->aCity;->aState)

vItemDest:= "" `nothing selected on the right (rows have been removed)

*evtUpdateText*(->vItem;->aCity;->aState) `update variable for left area with the new rows

**End if**

**End case**

\$0:=0 `event handled

### Handling Drag Action

The *evtRowsDragged* project method is called by the event callback method. It performs the following operations:

- remove the selected rows from the source area (the **aRows** array has been populated by the callback)
- add them at the top of the destination area (same order)
- redraw both areas
- select the new rows in the destination area

```
`evtRowsDragged
`updates both areas after a row(s) drag
C_LONGINT($1) `source AreaList Pro area
C_POINTER($2;$3) `-> source city array ; -> source state array
C_POINTER($4;$5) `-> destination city array ; -> source state array

INSERT ELEMENT ($4->;1;Size of array(aRows)) `insert rows at the beginning of destination arrays
INSERT ELEMENT ($5->;1;Size of array(aRows))

ARRAY LONGINT(aRowsToSelect;0) `deselect dragged rows in source area
AL_SetSelect($1;aRowsToSelect) `deselect source rows

ARRAY LONGINT(aRowsToSelect;Size of array(aRows)) `select dragged rows in destination area
For($i;Size of array(aRows);1;-1) `look backwards at each row selected by user
  `(aRows populated by event callback)
  $4->{$i}:=$2->{aRows{$i}} `city
  $5->{$i}:=$3->{aRows{$i}} `state
  DELETE ELEMENT ($2->;aRows{$i}) `delete source city
  DELETE ELEMENT ($3->;aRows{$i}) `delete source state
  aRowsToSelect{$i}:=$i
End for
AL_UpdateArrays($1;AL Recalculate arrays) `update source area
AL_GetDrgArea($1;eDestination;0) `destination area
AL_UpdateArrays(eDestination;AL Recalculate arrays) `update destination area
AL_SetSelect(eDestination;aRowsToSelect) `select new rows
$OK:=AL_GetSelect(eDestination;aRows) `get the rows for evtUpdateText
```

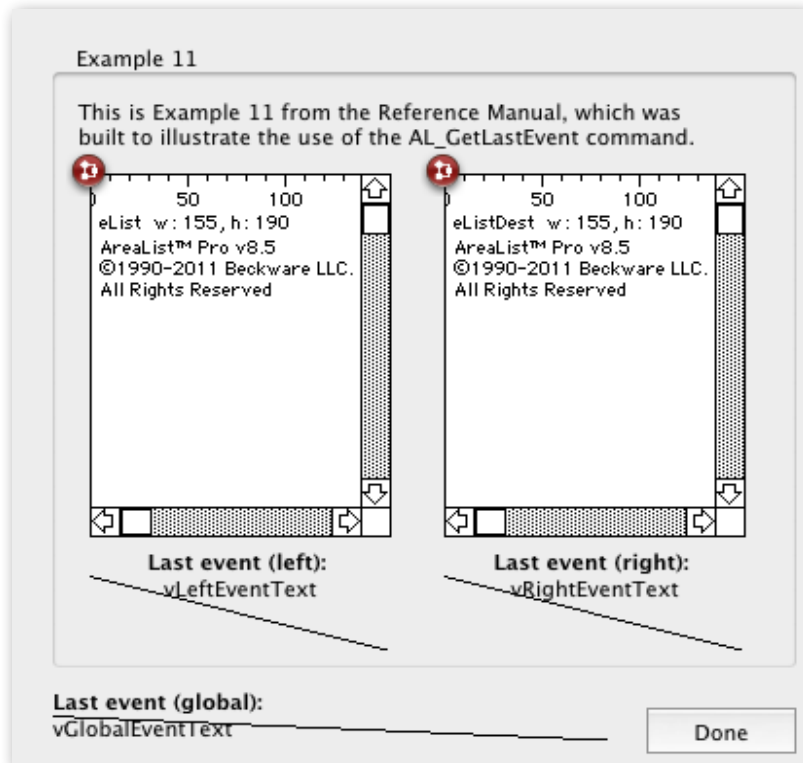
*evtUpdateText* is then called by the callback method to update the variable.



### Example 11 — Getting the Last Event in each Area

This example will demonstrate the use of [AL\\_GetLastEvent](#) in two different AreaList Pro areas.

Each area's last event is kept track of and displayed below the area. In addition, a text area on the bottom of the layout displays the last detected event in any area (formerly `ALProEvt`):



Our layout method takes care of the "Last event (global)" text (last event detected in any AreaList Pro area):

#### Case of

: (**Form event=On Load**)

vGlobalEventText:=""

: (**Form event=On Plug in Area**) `call from any of the two areas

AlpEventText (**AL\_GetLastEvent** ;>vGlobalEventText)

#### End case

## Examples

The *AlpEventText* project method returns the text value, based upon our event case of example (see [Determining the User's Action on an AreaList Pro Object](#)):

**C\_LONGINT**(\$1) `event code

**C\_POINTER**(\$2) `to the event description (text)

### Case of

```
: ($1=1)
    $2->:="Single-click"
: ($1=2)
    $2->:="Double-click"
: ($1=3)
    $2->:="Single-click in an empty part of the area (without displayed data)"
: ($1=4)
    $2->:="Double-click in an empty part of the area (without displayed data)"
: ($1=5)
    $2->:="Control-click (or right mouse click)"
: ($1=6)
    $2->:="Control-click (or right mouse click) in an empty part of the area (without displayed data)"
: ($1=7)
    $2->:="Vertical scroll"
: ($1=18)
    $2->:="Mouse has been moved (callback method only)"
: ($1=-1)
    $2->:="Sort button"
: ($1=-2)
    $2->:="Edit menu Select All"
: ($1=-3)
    $2->:="Column resized"
: ($1=-4)
    $2->:="Column lock changed"
: ($1=-5)
    $2->:="Row has been dragged from this area"
: ($1=-6)
    $2->:="User has invoked AreaList Pro Sort Editor"
: ($1=-7)
    $2->:="Column has been dragged from this area"
```

## Examples

```
: ($1=8)
  $2->:="Cell has been dragged from this area"
: ($1=9)
  $2->:="Object/window has been resized"
: ($1=10)
  $2->:="User clicked on column header, automatic sort won't be executed"
: ($1=11)
  $2->:="Control-click on column header"
: ($1=12)
  $2->:="Click on column footer"
: ($1=0) `No event, $2-> unchanged
```

### Else

```
$2->:="Unknown event"
```

### End case

```
If ($1#0) `Let's add the event code
```

```
$2->:=$2->+" (" + String($1) + ")"
```

### End if

Our eList object method On Plug in Area case updates the left area event text value:

### Case of

```
: (Form event=On Load) `initialize the AreaList Pro object
ALL RECORDS ([[Cities]]) `load all records in the Cities table
SELECTION TO ARRAY ([[Cities]City;aCity;[Cities]State;aState) `copy field values into arrays
$errorcode:=AL_SetArraysNam(Self->;1;2; "aCity";"aState") `display arrays in AreaList Pro object
AL_SetHeaders(Self->;1;2;"City";"State") `specify the values for the column headers
AL_SetRowOpts(Self->;AL Multiple row selection;AL Allow single or no row;2;1) `set multi-rows
mode, allow no selection, drag out, drag in
DEMO_Default(Self->)
vLeftEventText:=""
```

```
: (Form event=On Plug in Area) `call from any of the two areas
```

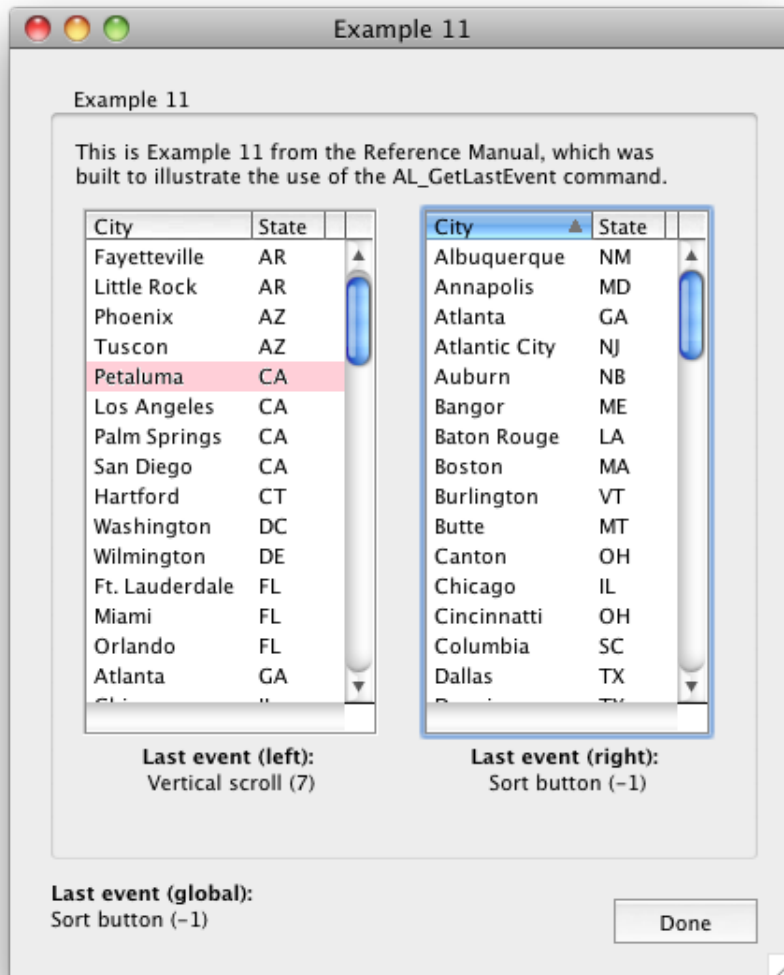
```
AlpEventText (AL_GetLastEvent (Self->);->vLeftEventText)
```

### End case

The eListDest object method performs a similar update with vRightEventText.

## Examples

Here is how the layout will look after a couple of events:



# AreaList Pro Constant List

## ALP Colors

Constant	Type	Value
AL White	S	white
AL Black	S	black
AL Magenta	S	magenta
AL Red	S	red
AL Cyan	S	cyan
AL Green	S	green
AL Blue	S	blue
AL Yellow	S	yellow
AL Gray	S	gray
AL Light gray	S	light gray
AL Use 4D palette color	S	

## ALP Patterns

Constant	Type	Value
AL White pattern	S	white
AL Black pattern	S	black
AL Gray pattern	S	gray
AL Light gray pattern	S	light gray
AL Dark gray pattern	S	dark gray

## ALP Events

Constant	Type	Value
AL Null event	L	0
AL Single click event	L	1
AL Double click event	L	2
AL Empty Area Single click	L	3
AL Empty Area Double click	L	4

## AreaList Pro Constant List

AL Single Control Click	L	5
AL Empty Area Control Click	L	6
AL Vertical Scroll Event	L	7
AL Mouse moved event	L	18
AL Sort button event	L	-1
AL Select all event	L	-2
AL Column resize event	L	-3
AL Column lock event	L	-4
AL Row drag event	L	-5
AL Sort editor event	L	-6
AL Column drag event	L	-7
AL Cell drag event	L	-8
AL Object resize event	L	-9
AL Column click event	L	-10
AL Column control click event	L	-11
AL Footer click event	L	-12

## ALP Entry callback actions

Constant	Type	Value
AL Click action	L	1
AL Tab key action	L	2
AL Shift_Tab key action	L	3
AL Return key action	L	4
AL Shift_Return key action	L	5
AL GotoCell action	L	6
AL ExitCell action	L	7
AL Cell validate action	L	8
AL SkipCell action	L	9
AL Other cell popup action	L	10
AL Active cell popup action	L	11

## ALP Array commands

Constant	Type	Value
AL No error in arrays	L	0
AL Not an array error	L	1
AL Wrong type array error	L	2
AL Wrong number rows error	L	3
AL Max arrays exceeded error	L	4
AL Low memory array error	L	5
AL Recalculate arrays	L	-2
AL Refresh and update arrays	L	-1

## ALP Sort commands

Constant	Type	Value
AL Sort in during off	L	0
AL Sort in during on	L	1
AL User sort off	L	0
AL User sort on	L	1
AL User sort bypass	L	2
AL User sort index only	L	3
AL Allow Sort editor off	L	0
AL Allow Sort editor on	L	1
AL Show Sort order off	L	0
AL Show Sort order on	L	1
AL Show Sort direction off	L	0
AL Show Sort direction on	L	1
AL Sort editor other process	L	-1
AL Sort editor cancelled	L	0
AL Sort editor accepted	L	1

## ALP Column commands

Constant	Type	Value
AL Apply to all columns	L	0
AL Auto width	L	0
AL Truncated upper left	L	0
AL Truncated centered	L	1
AL Scaled to fit	L	2
AL Scaled proportional	L	3
AL Just default	L	0
AL Just left	L	1
AL Just center	L	2
AL Just right	L	3
AL Use picture height off	L	0
AL Use picture height on	L	1
AL Allow column resize off	L	0
AL Allow column resize on	L	1
AL Resize in during off	L	0
AL Resize in during on	L	1
AL Allow column lock off	L	0
AL Allow column lock on	L	1
AL Display pixel width off	L	0
AL Display pixel width on	L	1
AL Use PICT Resource	L	65536
AL Use PicRef	L	131072

## ALP Row commands

Constant	Type	Value
AL Apply to all rows	L	0
AL Single row selection	L	0
AL Multiple row selection	L	1
AL Allow single row only	L	0
AL Allow single or no row	L	1
AL Move row info off	L	0
AL Move row info on	L	1
AL Enable row highlight	L	0



AL Disable row highlight	L	1
AL Remove row style	L	-1
AL Row style no change	L	256
AL Remove row font	S	-1
AL Row font no change	S	
AL Get row select low memory	L	0
AL Get row select succeeded	L	1

## ALP Entry commands

Constant	Type	Value
AL Column entry off	L	0
AL Column entry typed only	L	1
AL Column entry popup only	L	2
AL Column entry both	L	3
AL Disable spell check	L	0
AL Enable spell check	L	1
AL Entry_none select_sgl	L	0
AL Entry_none select_both	L	1
AL Entry_sgl select_none	L	2
AL Entry_dbl select_sgl	L	3
AL Entry_cmd_dbl select_both	L	4
AL Entry_shift_dbl select_both	L	5
AL Entry_opt_dbl select_both	L	6
AL Entry_ctrl_dbl select_both	L	7
AL Allow return off	L	0
AL Allow return on	L	1
AL Display seconds off	L	0
AL Display seconds on	L	1
AL Arrows move insertion	L	0
AL Arrows move cell	L	1
AL Enter key ignore	L	0
AL Enter key map to Tab	L	1
AL Enter key map to return	L	2
AL Use old popup icon	L	0
AL Use new popup icon	L	1

## AreaList Pro Constant List

AL Checkbox without title	L	0
AL Checkbox with title	L	1
AL Radio buttons	L	2
AL Cell not modified	L	0
AL Cell modified	L	1

## ALP Misc commands

Constant	Type	Value
AL Registration failed	L	0
AL Registration passed	L	1
AL Show headers	L	0
AL Hide headers	L	1
AL No area selection	L	0
AL Wide border area selection	L	1
AL System 7 area selection	L	2
AL 3D frame area selection	L	3
AL Post key default	S	\
AL Hide footers	L	0
AL Show footers	L	1
AL Modern look off	L	0
AL Modern look on	L	1
AL Copy hidden columns off	L	0
AL Copy hidden columns on	L	1
AL Copy field delim default	S	
AL Copy record delim default	S	
AL Copy field wrap default	S	
AL Toggle vert scroll bar	L	-1
AL Show vert scroll bar	L	-2
AL Hide vert scroll bar	L	-3
AL Toggle horz scroll bar	L	-1
AL Show horz scroll bar	L	-2
AL Hide horz scroll bar	L	-3
AL Area above vert scroll bar	L	0
AL Area below vert scroll bar	L	1
AL Area left horz scroll bar	L	2

## AreaList Pro Constant List

AL Area right horz scroll bar	L	3
AL No column dividers	S	
AL No row dividers	S	
AL Disable object resize	L	0
AL Enable object resize	L	1

## ALP Cell commands

Constant	Type	Value
AL Row selection	L	0
AL Single cell selection	L	1
AL Multiple cell selection	L	2
AL Move cell info off	L	0
AL Move cell info on	L	1
AL Cell optimization default	L	1
AL Remove cell style	L	-1
AL Cell style no change	L	256
AL Cell style not set	L	-1
AL Remove cell font	S	-1
AL Cell font no change	S	
AL Cell font not set	S	-1
AL Cell fore color not set	L	-1
AL Cell back color not set	L	-1
AL Remove cell entry	L	-1
AL Cell entry off	L	0
AL Cell entry on	L	1
AL Cell entry not set	L	-1
AL Get cell select low memory	L	0
AL Get cell select succeeded	L	1

## ALP Drag commands

Constant	Type	Value
AL Drag row data type	L	1
AL Drag column data type	L	2
AL Drag cell data type	L	3
AL Drag row with no key	L	0
AL Drag row with option key	L	1
AL Scroll area size default	L	30
AL Single row dragging	L	0
AL Multiple row dragging	L	1
AL Drag between rows	L	0
AL Drag onto row	L	1

## ALP Field commands

Constant	Type	Value
AL No error in fields	L	0
AL Not a file error	L	1
AL Not a field error	L	2
AL Wrong type field error	L	3
AL Max fields exceeded error	L	4
AL Wrong 4D vers for fields	L	5
AL Low memory field error	L	6
AL Refresh fields	L	0
AL Refresh and update fields	L	1
AL Recalculate fields	L	2

## ALP Appearance Constants

Constant	Type	Value
AL Default Interface	L	0
AL Platinum Interface	L	1
AL Force OSX Interface	L	2
AL Force XP Interface	L	3
AL Force Vista Interface	L	4

## ALP Edit Menu Constants

Constant	Type	Value
AL Edit Menu Undo Bit	L	0
AL Edit Menu Redo Bit	L	1
AL Edit Menu Cut Bit	L	2
AL Edit Menu Copy Bit	L	3
AL Edit Menu Paste Bit	L	4
AL Edit Menu Clear Bit	L	5
AL Edit Menu Select All Bit	L	6
AL Edit Menu Entry Bit	L	15
AL Edit Menu Setup Bit	L	16
AL Edit Menu Handled Bit	L	17
AL Edit Menu Undo Mask	L	1
AL Edit Menu Redo Mask	L	2
AL Edit Menu Cut Mask	L	4
AL Edit Menu Copy Mask	L	8
AL Edit Menu Paste Mask	L	16
AL Edit Menu Clear Mask	L	32
AL Edit Menu Select All Mask	L	64
AL Edit Menu All Items Mask	L	127
AL Edit Menu Entry Mask	L	32768
AL Edit Menu Setup Mask	L	65536
AL Edit Menu Handled Mask	L	131072

## ALP Format/Style Constants

Constant	Type	Value
AL Format Integer	L	1
AL Format Longint	L	2
AL Format Real	L	3
AL Format Boolean	L	4
AL Format Date	L	5
AL Format Picture	L	6
AL Style Header	L	1
AL Style List	L	2
AL Style Footer	L	3

# AreaList Pro Command Reference

## Alphabetical

<a href="#">%AL_DropArea</a> .....	221
<a href="#">%AreaListPro</a> .....	64
<a href="#">AL_ExitCell</a> (areaRef:L) .....	189
<a href="#">AL_GetAreaName</a> (areaRef:L; areaName:S) .....	223
<a href="#">AL_GetArrayNames</a> (areaRef:L; resultArray:X; options:L) → resultCode:L .....	67
<a href="#">AL_GetCellColor</a> (areaRef:L; cellColumn:I; cellRow:L; 4dForeColor:I; 4dBackColor:I) .....	129
<a href="#">AL_GetCellEnter</a> (areaRef:L; cellColumn:I; cellRow:L; enterability:I) .....	180
<a href="#">AL_GetCellHigh</a> (areaRef:L; startPosition:I; endPosition:I) .....	183
<a href="#">AL_GetCellMod</a> (areaRef:L) → resultCode:L .....	181
<a href="#">AL_GetCellOpts</a> (areaRef:L; cellSelection:I; moveWithData:I; optimization:I) .....	96
<a href="#">AL_GetCellRGBColor</a> (areaRef:L; cellColumn:I; cellRow:L; cellForeRed:L; cellForeGreen:L; cellForeBlue:L; cellBackRed:L; cellBackGreen:L; cellBackBlue:L) .....	131
<a href="#">AL_GetCellSel</a> (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X) → resultCode:L .....	215
<a href="#">AL_GetCellStyle</a> (areaRef:L; cellColumn:I; cellRow:L; styleNum:I; fontName:S) .....	126
<a href="#">AL_GetCellText</a> (areaRef:L; text:T; flag:L) .....	218
<a href="#">AL_GetCellValue</a> (areaRef:L; cellRow:L; cellColumn:I; alphanumericData:T; pictData:P) .....	181
<a href="#">AL_GetClickedRow</a> (areaRef:L) → clickedRow:L .....	212
<a href="#">AL_GetColLock</a> (areaRef:L) → columns:L .....	216
<a href="#">AL_GetColOpts</a> (areaRef:L; allowColumnResize:I; automaticResize:I; allowColumnLock:I; hideLastColumns:I; displayPixelWidth:I; dragColumn:I; acceptDrag:I) .....	94
<a href="#">AL_GetColumn</a> (areaRef:L) → clickedColumn:L .....	211
<a href="#">AL_GetCopyOpts</a> (areaRef:L; includeHiddenCols:I; fieldDelimiter:S; recordDelimiter:S; fieldWrapper:S) .....	104
<a href="#">AL_GetCurrCell</a> (areaRef:L; cellColumn:I; cellRow:L) .....	187
<a href="#">AL_GetDrgArea</a> (areaRef:L; destArea:L; destProcessID:I) .....	199
<a href="#">AL_GetDrgDstCol</a> (areaRef:L; destCol:I) .....	202
<a href="#">AL_GetDrgDstRow</a> (areaRef:L; destRow:L) .....	202
<a href="#">AL_GetDrgDstTyp</a> (areaRef:L; destDataType:I) .....	200
<a href="#">AL_GetDrgSrcCol</a> (areaRef:L; sourceCol:I) .....	198
<a href="#">AL_GetDrgSrcRow</a> (areaRef:L; sourceRow:L) .....	197
<a href="#">AL_GetFields</a> (areaRef:L; tableArray:X; fieldArray:X) → resultCode:L .....	165
<a href="#">AL_GetFooters</a> (areaRef:L; footerList:X; options:L) → resultCode:L .....	76
<a href="#">AL_GetFormat</a> (areaRef:L; columnNumber:I; format:S; columnJust:I; headerJust:I; footerJust:I; usePictHeight:I) .....	82
<a href="#">AL_GetFtrStyle</a> (areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I) .....	85

## AreaList Pro Command Reference — Alphabetical

[AL\\_GetHdrStyle](#) (areaRef:L; columnNumber:L; fontName:S; size:I; styleNum:I)

[AL\\_GetHeaderOptions](#) (areaRef:L; options:L; iconRef:L; callbackMethod:S)

[AL\\_GetHeaders](#) (areaRef:L; headerList:X; options:L) → resultCode:L

[AL\\_GetLastEvent](#) (areaRef:L) → eventCode:L

[AL\\_GetLine](#) (areaRef:L) → selectedRow:L

[AL\\_GetMiscOpts](#) (areaRef:L; hideHeaders:I; areaSelected:I; postKey:S; showFooters:I; useModernLook:I)

[AL\\_GetMode](#) (areaRef:L) → resultCode:L

[AL\\_GetPictureEscape](#) (areaRef:L) → escapeChar:S

[AL\\_GetPluginPath](#) → path:S

[AL\\_GetPrevCell](#) (areaRef:L; cellColumn:I; cellRow:L)

[AL\\_GetRowOpts](#) (areaRef:L; multiRows:I; allowNoSelection:I; dragRow:I; acceptDrag:I; moveWithData:I; disableRowHighlight:I)

[AL\\_GetScroll](#) (areaRef:L; verticalScroll:L; horizontalScroll:I)

[AL\\_GetSelect](#) (areaRef:L; array:X) → resultCode:L

[AL\\_GetSort](#) (areaRef:L; column1:I; ...; columnN:I)

[AL\\_GetSortedCols](#) (areaRef:L; sortList:X) → resultCode:L

[AL\\_GetSortEditorParams](#) (areaRef:L; windowTitle:S; prompt:S; headerList:X; sortList:X) → resultCode:L

[AL\\_GetStyle](#) (areaRef:L; columnNumber:I; fontName:S; size:I; styleNum:I)

[AL\\_GetTable](#) (areaRef:L) → tableNumber:L

[AL\\_GetVersion](#) → version:S

[AL\\_GetWidths](#) (areaRef:L; columnNumber:I; numWidths:I; width1:I; ...; widthN:I)

[AL\\_GotoCell](#) (areaRef:L; cellColumn:I; cellRow:L)

[AL\\_InsArrayNam](#) (areaRef:L; columnNumber:I; numArrays:I; array1:S; ...; arrayN:S) → resultCode:L

[AL\\_InsertFields](#) (areaRef:L; tableNum:I; columnNumber:I; numFields:I; fieldNum1:I ... fieldNumN:I) → resultCode:L

[AL\\_Register](#) (registrationKey:S) → resultCode:L

[AL\\_RemoveArrays](#) (areaRef:L; columnNumber:I; numArrays:I)

[AL\\_RemoveFields](#) (areaRef:L; columnNumber:I; numFields:I)

[AL\\_SetAltRowClr](#) (areaRef:L; alpRowBackColor:S; 4dRowBackColor:I; options:L)

[AL\\_SetAltRowColor](#) (areaRef:L; red:L; green:L; blue:L; options:L)

[AL\\_SetAreaName](#) (areaRef:L; areaName:S)

[AL\\_SetArraysNam](#) (areaRef:L; columnNumber:I; numArrays:I; array1:S; ...; arrayN:S) → resultCode:L

[AL\\_SetBackColor](#) (areaRef:L; columnNumber:I; alpHdrBackColor:S; 4dHdrBackColor:I;  
alpListBackColor:S; 4dListBackColor:I; alpFtrBackColor:S; 4dFtrBackColor:I)

[AL\\_SetBackRGBColor](#) (areaRef:L; columnNumber:L; hdrBackRed:L; hdrBackGreen:L; hdrBackBlue:L;  
listBackRed:L; listBackGreen:L; listBackBlue:L; ftrBackRed:L; ftrBackGreen:L; ftrBackBlue:L)

[AL\\_SetCalcCall](#) (areaRef:L; columnNumber:I; calcCallback:S)

[AL\\_SetCallbacks](#) (areaRef:L; entryStartedMethod:S; entryFinishedMethod:S)

## AreaList Pro Command Reference — Alphabetical

[AL\\_SetCellBorder](#) (areaRef:L; cellColumn:I; cellRow:L; borderLeft:I; borderTop:I; borderRight:I; borderBottom:I; offset:I; width:F; redColor:I; greenColor:I; blueColor:I)

[AL\\_SetCellColor](#) (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; alpForeColor:S; 4dForeColor:I; alpBackColor:S; 4dBackColor:I)

[AL\\_SetCellEnter](#) (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; enterability:I)

[AL\\_SetCellFrame](#) (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; offset:I; width:F; redLightColor:I; greenLightColor:I; blueLightColor:I; redDarkColor:I; greenDarkColor:I; blueDarkColor:I; clearAllBorders:I)

[AL\\_SetCellHigh](#) (areaRef:L; startPosition:I; endPosition:I)

[AL\\_SetCellIcon](#) (areaRef:L; cellColumn:I; cellRow:L; pictRef:P; iconAlignment:I; horPosition:I; vertPosition:I; offset:I; scaling:I)

[AL\\_SetCellOpts](#) (areaRef:L; cellSelection:I; moveWithData:I; optimization:I)

[AL\\_SetCellRGBColor](#) (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; cellForeColor:L; cellForeGreen:L; cellForeBlue:L; cellBackColor:L; cellBackGreen:L; cellBackBlue:L)

[AL\\_SetCellSel](#) (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X)

[AL\\_SetCellStyle](#) (areaRef:L; firstCellCol:I; firstCellRow:L; lastCellCol:I; lastCellRow:L; cellArray:X; styleNum:I; fontName:S)

[AL\\_SetCellText](#) (areaRef:L; text:T; flag:L)

[AL\\_SetCellValue](#) (areaRef:L; row:L; column:I; alphaNumericData:S; pictData:P)

[AL\\_SetColLock](#) (areaRef:L; columns:I)

[AL\\_SetColOpts](#) (areaRef:L; allowColumnResize:I; automaticResize:I; allowColumnLock:I; hideLastColumns:I; displayPixelWidth:I; dragColumn:I; acceptDrag:I)

[AL\\_SetCopyOpts](#) (areaRef:L; includeHiddenCols:I; fieldDelimiter:S; recordDelimiter:S; fieldWrapper:S)

[AL\\_SetDefaultFormat](#) (selector:L; format:S)

[AL\\_SetDefaultStyle](#) (selector:L; fontName:S; size:L; styleNum:L)

[AL\\_SetDividers](#) (areaRef:L; colDividerPattern:S; alpColDividerColor:S; 4dColDividerColor:I; rowDividerPattern:S; alpRowDividerColor:S; 4dRowDividerColor:I)

[AL\\_SetDrgDst](#) (areaRef:L; destDataType:I; dstCode1:S; ...; dstCode10:S)

[AL\\_SetDrgOpts](#) (areaRef:L; dragRowWithOptKey:I; scrollAreaSize:I; multiRowDrag:I; dragOntoRow:I)

[AL\\_SetDrgSrc](#) (areaRef:L; sourceDataType:I; srcCode1:S; ...; srcCode10:S)

[AL\\_SetDropDst](#) (dropAreaRef:L; dstCode1:S; ...; dstCodeN:S)

[AL\\_SetEditMenuCallback](#) (areaRef:L; callbackMethod:S) → resultCode:L

[AL\\_SetEnterable](#) (areaRef:L; columnNumber:I; enterability:I; popupArray:X; menuPackRef:L)

[AL\\_SetEntryCtrls](#) (areaRef:L; columnNumber:I; controlType:I)

[AL\\_SetEntryOpts](#) (areaRef:L; entryMode:I; allowReturn:I; displaySeconds:I; moveWithArrows:I; mapEnterKey:I; decimalCharForWin:S; useNewPopUpIcon:I)

[AL\\_SetEventCallback](#) (areaRef:L; callbackMethod:S; flag:L) → resultCode:L

[AL\\_SetFields](#) (areaRef:L; tableNum:I; columnNumber:I; numFields:I; fieldNum1; ...; fieldNumN:I) → resultCode:L

[AL\\_SetFile](#) (areaRef:L; tableNum:I) → resultCode:L

[AL\\_SetFilter](#) (areaRef:L; columnNumber:I; entryFilter:S)

[AL\\_SetFooters](#) (areaRef:L; columnNumber:I; numFooters:I; footer1:S; ...; footerN:S)

[AL\\_SetForeColor](#) (areaRef:L; columnNumber:I; alpHdrForeColor:S; 4dHdrForeColor:I; alpListForeColor:S; 4dListForeColor:I; alpFtrForeColor:S; 4dFtrForeColor:I)



# AreaList Pro Command Reference — Alphabetical

[AL\\_SetForeColor](#) (areaRef:L; columnNumber:L; hdrForeRed:L; hdrForeGreen:L; hdrForeBlue:L; listForeRed:L; listForeGreen:L; listForeBlue:L; ftrForeRed:L; ftrForeGreen:L; ftrForeBlue:L)

[AL\\_SetFormat](#) (areaRef:L; columnNumber:L; format:S; columnJust:L; headerJust:L; footerJust:L; usePictHeight:L)

[AL\\_SetFtrStyle](#) (areaRef:L; columnNumber:L; fontName:S; size:L; styleNum:L)

[AL\\_SetHdrStyle](#) (areaRef:L; columnNumber:L; fontName:S; size:L; styleNum:L)

[AL\\_SetHeaderIcon](#) (areaRef:L; columnNumber:L; iconAlignment:L picture:P; horPosition:L; vertPosition:L; offset:L; scaling:L)

[AL\\_SetHeaderOptions](#) (areaRef:L; options:L; iconRef:L; callbackMethod:S)

[AL\\_SetHeaders](#) (areaRef:L; columnNumber:L; numHeaders:L; header1:S; ...; headerN:S)

[AL\\_SetHeight](#) (areaRef:L; numHeaderLines:L; headerHeightPad:L; numRowsLines:L; rowHeightPad:L; numFooterLines:L; footerHeightPad:L)

[AL\\_SetInterface](#) (areaRef:L; appearance:L; sortIndicator:L; useEllipsis:L; ignoreMenuMeta:L; clickDelay:L; allowPartialRow:L; useOldPopup:L; entryControls: L)

[AL\\_SetLine](#) (areaRef:L; rowNumber:L)

[AL\\_SetMainCalls](#) (areaRef:L; areaEnteredMethod:S; areaExitedMethod:S)

[AL\\_SetMinRowHeight](#) (areaRef:L; minRowHeight:L)

[AL\\_SetMiscColor](#) (areaRef:L; selector:L; alpColor:S; 4dColor:L)

[AL\\_SetMiscOpts](#) (areaRef:L; hideHeaders:L; areaSelected:L; postKey:S; showFooters:L; useModernLook:L)

[AL\\_SetMiscRGBColor](#) (areaRef:L; selector:L; red:L; green:L; blue:L)

[AL\\_SetPictureEscape](#) (areaRef:L; escapeChar:S)

[AL\\_SetRGBDividers](#) (areaRef:L; colDividerPattern:S; colDividerRed:L; colDividerGreen:L; colDividerBlue:L; rowDividerPattern:S; rowDividerRed:L; rowDividerGreen:L; rowDividerBlue:L)

[AL\\_SetRowColor](#) (areaRef:L; rowNumber:L; alpRowForeColor:S; 4dRowForeColor:L; alpRowBackColor:S; 4dRowBackColor:L)

[AL\\_SetRowOpts](#) (areaRef:L; multiRows:L; allowNoSelection:L; dragRow:L; acceptDrag:L; moveWithData:L; disableRowHighlight:L)

[AL\\_SetRowRGBColor](#) (areaRef:L; rowNumber:L; rowForeRed:L; rowForeGreen:L; rowForeBlue:L; rowBackRed:L; rowBackGreen:L; rowBackBlue:L)

[AL\\_SetRowStyle](#) (areaRef:L; rowNumber:L; styleNum:L; fontName:S)

[AL\\_SetScroll](#) (areaRef:L; verticalScroll:L; horizontalScroll:L)

[AL\\_SetSelect](#) (areaRef:L; rowsToSelect:X)

[AL\\_SetSort](#) (areaRef:L; column1:L; ...; columnN:L)

[AL\\_SetSortedCols](#) (areaRef:L; sortList:X) → resultCode:L

[AL\\_SetSortEditorParams](#) (areaRef:L; windowTitle:S; prompt:S; labelList:X; columnNumberList:X) → resultCode:L

[AL\\_SetSortOpts](#) (areaRef:L; automaticSort:L; userSort:L; allowSortEditor:L; sortEditorPrompt:S; showSortOrder:L; showSortDirIndicator:L)

[AL\\_SetStyle](#) (areaRef:L; columnNumber:L; fontName:S; size:L; styleNum:L)

[AL\\_SetSubSelect](#) (areaRef:L; firstRecord:L; numRecords:L)

[AL\\_SetWidths](#) (areaRef:L; columnNumber:L; numWidths:L; width1:L; ...; widthN:L)

[AL\\_ShowSortEd](#) (areaRef:L) → sortDone:L

[AL\\_SkipCell](#) (areaRef:L)

[AL\\_UpdateArrays](#) (areaRef:L; updateMethod:L)

[AL\\_UpdateFields](#) (areaRef:L; updateMethod:L)